LOGO

תכנות פונקציונלי

ש. מנולה

LOGO
LOGO
LOGO
$f(x) =$ LOGO
LOGO

ערך י. עמיהוד

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי

LOGO

תכנות פונקציונלי

לתלמיד למופת

ברכת כל טוב

2/89

לבני משפחתי

באהבה

עיבוד תמלילים ועריכה לשונית: שרה עמיהוד

LOGO

תכנות פונקציונלי

ש. מנולה

ערך י. עמיהוד

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד־עמי

L O G O

Functional Programming

by S. Manola

edited by I. Amihud

הוצאת הוד-עמי
לספרי מחשבים בע"מ

ת.ד. 560, רמת-גן 52105
טל המשרד: 052-541207

הודפס בישראל
שבט תשמ"ט, פברואר 1989

Hod-Ami Ltd
P.O.B. 560 Ramat-Gan
Israel, February 1989

ISBN 965-361-003-1 מסת"ב

הקדמה

שפת לוגו (LOGO) הינה שפת תכנות מתקדמת ומתוחכמת. זוהי גירסה ידידותית של שפת עיבוד הרשימות המתקדמת LISP - שפת הבינה המלאכותית. לוגו בנויה על פי עקרונות התכנות המבני (מודולרי), המאפשרים ראיית השלם כמורכב מחלקים עצמאיים - מבנה של פרוצדורות ופונקציות. עקרונות אלה מנחים את המתכנת למשמעת עבודה וסגנון תכנות נכון, לשם פיתוח של תכניות מורכבות, אשר ניתן גם לפקח ולשלוט בהן ביעילות.

לוגו הינה שפה בעלת אופי פונקציונלי ועל-כן עוסק הספר בעיקר בתכנות הפונקציונלי. הרכבת פונקציות הינה אמצעי עיקרי ליצירת ביטויים מורכבים בתכנות, ולשם כך נדרשות דרכי חשיבה חדשות ומתקדמות. מסיבה זו נמנעתי מלכלול הוראת הצבה להגדרת משתנה גלובלי כמו 'MAKE', כל עוד ניתן לוותר עליה. יתרה מזו, התכניות בלעדי 'MAKE' נכתבות באופן פשוט ואלגנטי יותר.

במהלך התיאור ולימוד השפה הצגתי בפני הקורא מושגים רבים מעולם מדעי המחשב, כמו משתנים, הליכים, פונקציות ועוד. הלומד יוכל להכיר את מבני הפיקוח השונים המוכרים בפתרון בעיות חישוביות (בעיות אלגוריתמיות), כמו המבנה הסידרתי, המבנה המפוצל, מבנה הלולאה, השיגרה ובעיקר - את המבנה הרקורסיבי.

אין לי כל כוונה ללמד נושאים במתמטיקה, כמו תורת הקבוצות למשל, וגם לא נושאים במבנה המחשב כמו שיטות ספירה. לכן, יש להתייחס לסעיפים המכילים תרגילים העוסקים בנושאים אלה, כמיועדים לתלמיד אשר רכש ממקורות אחרים את הידע הבסיסי הדרוש.

בספר ניתנים תרגילים רבים לדוגמה, הכוללים פתרונות מפורטים, הסברים ותיאור של שלבי פתרון. הם אינם מיועדים לחיקוי עיוור, אלא מכוונים להצגת דרכי חשיבה שיטתית ותהליכי עבודה מובנים לבניית הפתרון/התכנית. בכל פרק מוצגים התרגילים ברמת קושי עולה בהתאם להתקדמות הלימוד.

ספר זה מתאים בעיקר לשפת IBM LOGO. מכיון שההוראות הנלמדות בו שייכות לגירסה של LCSI (Logo Computer Systems Inc), הוא מתאים גם למשתמשי הגירסה של APPLE LOGO II וגם למשתמשי הגירסה של APPLE LOGO. כללי התחביר והדוגמאות מתאימים לכל אחד מהם. במקום שקיימים הבדלים בין הגירסאות (ואלה מעטים בלבד), ניתנים לנוחות הקורא הערות, הסברים מפורטים והנחיות. בנוסף ניתן תיאור של הגירסה של MIT/TERRAPIN, המתאימה גם למחשבי קומודור (COMMODORE), כדי שגם משתמשי גירסה זו יוכלו להשתמש בספר.

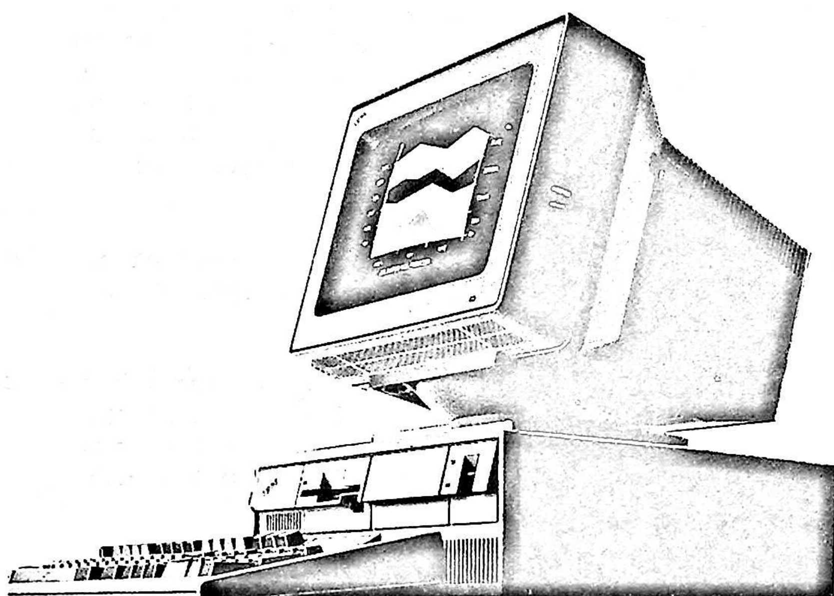
תכנות פונקציונלי בשפת LOGO מיועד לתלמידים בבתי ספר תיכוניים ובאוניברסיטאות, ולכל מי שמעוניין ללמוד ולהכיר שפה לוגית מתקדמת. הוא מכוון אל כל מי שמעדיף להתקדם בצעדים בטוחים באופן מבוקר, להגביר את הקצב ולקלוט נושאים חדשים. צעדי הצב של שפת LOGO טובים ואף מומלצים למתחילים, אשר לומדים את העקרונות של שפת התכנות והגרפיקה ואינם בתחום העיסוק של ספר זה.

תודתי נתונה לפרופ' יהושפט גבעון, שעזר לי בהערותיו, הדריך אותי בעצותיו, גילה לי מרעיונותיו ועודד אותי בהוצאת ספר זה לאור.

אני מקווה שספר זה יעורר את התעניינותו של הלומד לחיפוש אחר אתגרים חדשים, ויעניק למשתמש בו את מלוא ההנאה שבהתלהבות היצירה.

ש. מנולה

תוכן העניינים



- IBM PC LOGO
- APPLE LOGO
- APLLE LOGO II

גירסאות לוגו אלו פותחו ע"י חברת
.LCSI - LOGO Computer Systems Inc

פרק 1

WELCOME TO LOGO

זוהי "קבלת הפנים" המופיעה על המסך, בה שפת לוגו מברכת אותנו, כאשר מטעינים אותה לזיכרון המחשב.

בתחילת שורה חדשה מופיע התו 'סימן שאלה' (?). זוהי התווית של שפת לוגו, או סימן ההיכר של השפה, הקרוי באנגלית Prompt, ולידו מופיע ריבוע מהבהב (זהו **הסמן**, Cursor). הסימן '?' מציין שלוגו מוכנה לקבל הוראות מאוצר הפקודות השייכות לה.

הוראת ההדפסה PRINT

PRINT היא הוראת פלט המשמשת להדפסת **דבר** (Thing) כלשהו על המסך. **דבר** יכול להיות תו, מילה, מספר, משפט (רשימה של מילים), ערך של ביטוי חשבוני, ערך של משתנה ועוד.

מבנה המשפט: PRINT **דבר**

או בקיצור: PR **דבר**

יש להפריד בין המילה PRINT לבין המשך המשפט על-ידי **תו-רווח**. **תו-הרווח בלוגו משמש להבחנה בין חלקי המשפט של השפה.** בסוף מתן ההוראה יש ללחוץ על המקש <RETURN> (או <RET> בקיצור), ובכך להודיע למחשב שסיימנו לכתוב את הפקודה ועליו לבצעה כעת. יש מחשבים בהם שם מקש זה הוא <ENTER>, ויש אחרים שבהם המקש מסומן על-ידי חץ שמאלי בדמוי וו (↵).

הדפסת נתון מספרי

אם נכתוב 'PR 5' ונלחץ על <RETURN> --->	?PR 5
המחשב ידפיס את המספר:	5
עוד דוגמה --->	?PR 1988
המחשב ידפיס:	1988
	?

בגמר ההדפסה מופיעה שוב התווית של לוגו (סימן השאלה '?') בראש שורה חדשה, ולידה הסמן המהבהב המציין שלוגו מוכנה לקבל הוראות חדשות.

יש לזכור, שכל עוד לא הקשנו על <RET>, לא תהיה שום תגובה מן המחשב, כי אין הוא יודע אם סיימנו לכתוב את ההוראה או לא, אם רצינו לכתוב 5 או 540.

בהוראה PR מתייחס המחשב לערכו של הנתון המספרי ולא לסיפרה המסמלת אותו. כך הדבר גם לגבי הפעולות החשבוניות. אין המחשב מתייחס אל התו המסמל את פעולת החיבור '+' למשל, אלא למשמעות הפעולה עצמה והוא מבצע אותה בהתאם. לכן נוכל לבקש מהמחשב להדפיס לנו את ערכו של ביטוי חשבוני.

כאשר נכתוב --->	?PR 5+4
המחשב ידפיס:	9
	?

כאן המחשב לא הדפיס את הסיפרה 5, את פעולת החיבור '+' ואת הסיפרה 4 כפי שהצגנו זאת, אלא הוא חישב את ערך הביטוי '5+4' והדפיס את התוצאה.

אם היינו כותבים למחשב 5+4 בהשמטת ההוראה PR, היינו מקבלים הערה כגון:

================================

I DON'T KNOW WHAT TO DO WITH 9

================================

מכיון שהמחשב מתייחס אל התוצאה כאל ערך, יש להורות לו מה לעשות בערך זה - האם לאחסן אותו, האם לערוך בו חישובים נוספים, האם להדפיס אותו. ההוראה PR, היא אחת הדרכים לפנייה למחשב.

הדפסת מילה

מילה יכולה להיות מורכבת מתו אחד או יותר. על מנת לגרום למחשב להתייחס למילה כאוסף של תווים ולא אל המשמעות שלה, יש להדפיס גרשיים (") בתחילת המילה.

אם נכתוב ---> ?PR "PRINT

תופיע על המסך המילה: PRINT

?

כאן מדפיס המחשב את רצף התווים המרכיבים את המילה 'PRINT', מבלי לפרש את המשמעות שלה ומבלי לקבל אותה כהוראה שעליו לבצע.

נוכל לבקש לכתוב כל מילה שנבחר, למשל: ?PR "SHALOM

על המסך תופיע המילה: SHALOM

?

הגרשיים מציינים את תחילתה של מילה, וסופה מצוין על-ידי תו-הרווח, כפי שמציינים סוף של כל נתון בלוגו (ולא על-ידי גרשיים כפי שנהוג בשפות אחרות).

אם נכתוב ---> ?PR "SHALOM"

המחשב ידפיס: SHALOM"

?

הגרשיים שבסוף המילה SHALOM הם חלק בלתי נפרד ממנה, דבר שלא התכוונו אליו.

הוראת PR פועלת על דבר אחד בלבד. מילה נחשבת לדבר אחד, אך שתי מילים הן שני דברים.

אם נכתוב <---   ?PR "GOOD "MORNING

המחשב ידפיס: GOOD

אך יעיר לגבי המילה 'MORNING':

```
--------------------------------------
I DON'T KNOW WHAT TO DO WITH MORNING
--------------------------------------
```

על הדרך לכתוב מילים אחדות בשורה אחת נלמד בהמשך.

עוד דוגמה <---

```
?PR 15 23
15
I DON'T KNOW WHAT TO DO WITH 23
?
```

בלוגו ניתן לכתוב יותר מהוראה אחת בשורה. כלומר, הקשת <RET> באה אחרי כתיבת סידרה של הוראות בזו אחר זו, כל עוד מקפידים על הרווחים במקומות הנכונים.

אם נכתוב <---

יופיע על המסך הפלט הבא:

```
?PR "HAPPY PR "NEW PR "YEAR
HAPPY
NEW
YEAR
?
```

כפי שרואים, בכל אחת מהוראות PR עובר הסמן בתום ההדפסה לתחילתה של שורה חדשה, ושם הוא מדפיס את הנתון הבא.

כיצד, אם כן, ניתן להדפיס יותר ממילה אחת באותה השורה?

הדפסת רשימה

הדרך להדפיס מספר מילים, מספרים וכו', היא לתחום את כולם בתוך רשימה אחת בין שני סוגריים **מרובעים**, אחד שמאלי '[' ואחד ימני ']'. כאשר נוהגים כך, אין צורך להקדים כל מילה בגרשיים, או לכתוב PR לפני כל נתון.

לדוגמה ---> ?PR [GOOD MORNING]

נקבל: GOOD MORNING

דוגמה נוספת ---> ?PR [HAPPY NEW YEAR]

נקבל: HAPPY NEW YEAR

ועוד דוגמה ---> ?PR [15 23 48]

נקבל: 15 23 48

אוסף של נתונים שתוחמים אותם ב**סוגריים מרובעים** נקרא **רשימה.** גם כאן PR פועלת על דבר אחד, והפעם זוהי רשימה.

יש לציין שכל נתון הנמצא בתוך רשימה במשפט ההוראה PR, הינו חסר משמעות לגבי המחשב, ולכן הוא יודפס כפי שהוצג.

אם נכתוב ---> ?PR [45 * 3]

המחשב ידפיס: 45 * 3

?

המחשב לא חישב את ערכו של הביטוי החשבוני, ולכן לא הודפסה התוצאה, אלא הופיעו על המסך הנתונים שברשימה.

הערה: בדוגמאות שלמעלה סומנו הסוגריים המרובעים בהדגשה. הדבר נעשה לצורך הלימוד בלבד, ואין זה חלק מצורת הכתיבה בלוגו. בהמשך נסמן לעיתים קרובות הוראות, או נדגיש לשם הבלטה, לשם לימוד בלבד.

הדפסת מספר דברים

הדרך להדפיס מספר דברים בשורה היא לתחום את **משפט ההוראה** בין שני סוגריים **עגולים.** כלומר, יש להקדים את הפקודה PR בסוגר שמאלי '(', ולסיים את המשפט בסוגר ימני ')'.

המבנה הכללי:

```
---------------------------------
(PR  1דבר  2דבר  ...  דברת)
---------------------------------
```

לדוגמה --->

על המסך יופיע המשפט:

```
?(PR "HAPPY [NEW YEAR] 2000)
HAPPY NEW YEAR 2000
?
```

נסה לגלות מה ידפיס המחשב כאשר נרשום את ההוראה הבאה:

```
?(PR  [238 + 420 =]  238 + 420)
```

שורת הוראה

מספר התווים בשורה בלוגו הוא 128. זוהי שורת הוראה לוגית הנקלטת על-ידי המחשב, ולא שורה פיזית, שהיא לא יותר מ-40 או 80 תווים. כאשר מזינים מספר רב של דברים, ללא הקשה על <RET> בעת כתיבת הוראה ומגיעים לסוף השורה הפיזית, מופיע במקום ה-40 חץ ימינה '<--' בגירסה של IBM LOGO, או סימן קריאה '!' בגירסה של APPLE LOGO II ושל APPLE LOGO. זהו ציון לכך שאין זה סוף **שורת הוראה**. הסמן עובר לשורה פיזית חדשה, כדי לאפשר לנו לרשום את המשך המשפט.

רצוי לבדוק לפני כל לחיצה על <RET>, אם מה שכתבת זה אכן מה שרצית לכתוב. למשל, אם לא כתבת PPR במקום PR, אולי שכחת רווח במקום שצריך ורשמת PR"HELLO במקום PR "HELLO, או אם חסר סוגר מרובע או עגול במקום המתאים. ניתן לערוך תיקונים כל עוד הסמן נמצא באותה שורת הוראה, ולא עבר לתחילת שורה חדשה.

- ב-IBM LOGO התיקון נעשה בעזרת מקש החץ לאחור (Back-arrow), אשר בכל הקשה מוחק תו אחד משמאל לסמן. בדרך זו ניתן למחוק תווים לא רצויים ולכתוב אחרים.
- ב-APPLE LOGO II מחיקת תו שמשמאל לסמן נעשית באמצעות מקש DEL.

- ב-APPLE LOGO מבצע תפקיד זה החץ השמאלי '-->'.

דרך אחרת לתיקון, היא על-ידי הזזת הסמן עד לאיזור השגיאה:
- ב-IBM LOGO וב-APPLE LOGO II נעשה הדבר באמצעות החץ השמאלי, אשר מניע את הסמן לתו שמשמאלו מבלי למחוק אותו. כך ניתן להגיע לאיזור השגיאה, לתקן את הנדרש, להוסיף תו שחסר וכו'.
- ב-APPLE LOGO יש ללחוץ על המקש <CTRL>, ובעודו לחוץ להקיש את האות 'B' (BACK).

הכתיבה בלוגו נעשית בשיטת PUSH-MODE או INSERT-MODE. אין חשש שתו חדש "יעלה" על תו קיים, מפני שכל השורה מן הסמן והלאה זזה ימינה ומפנה מקום לתו שהוכנס. לאחר התיקון יש להניע את הסמן ימינה לסוף השורה, בעזרת חץ ימין '<--', על מנת להמשיך היכן שהפסקנו.

אם אחרי שלחצת על <RET>, הודיע לך המחשב הערה כתוצאה משגיאת כתיב, ניתן לחסוך בכתיבת כל המשפט מחדש. השתמש בפעולה המעתיקה את המשפט האחרון שנכתב:
- ב-IBM LOGO הדבר נעשה על ידי הקשת <F3>.
- ב-APPLE LOGO II יש ללחוץ על <CTRL> ובעודו לחוץ להקיש על מקש האות 'R'.
- וב-APPLE LOGO - על-ידי לחיצה על <CTRL> ועל מקש האות 'Y'.

פעולה זו תעתיק על המסך את כל השורה שנכתבה לפני הקשת ה-<RET> האחרון, כשהסמן מופיע בסופה. במצב זה ניתן לערוך את התיקון הדרוש כפי שהוסבר לעיל.

דילוג שורה

אם תרצה שהמחשב ידלג על שורה מבלי להדפיס בה דבר, ותרשום את הפקודה PR ללא דבר כלשהו להדפסה ---> ?PR
תקבל את ההודעה הבאה בגירסת IBM LOGO:

```
--------------------------------
NOT ENOUGH INPUTS TO PRINT
--------------------------------
```

אנו לומדים מכך, שהוראת PR בלוגו דורשת קלט, אשר כולל דבר אחד או דברים אחדים.

הערה: ב-IBM LOGO, ב-APPLE LOGO II וב-APPLE LOGO יש הודעות שגיאה שונות במעט זו מזו.

איך נוכל, אם כן, לדלג על שורה, להשאיר שורת רווח ולעבור לשורה חדשה? זאת נעשה

א. באמצעות הרשימה הריקה '[]' (= הסוגריים המרובעים, כאשר לא מופיע ביניהם כל תו).

ב. באמצעות המילה הריקה '"' (= גרשיים, כאשר אף לא תו אחד רשום אחריהם).

נוכל לכתוב כך ---> ?PR []

או כך ---> ?PR "

בכל אחד משני המקרים יעבור הסמן לתחילת שורה חדשה מבלי להדפיס דבר.

הדפסת רווח

תו-הרווח משמש בלוגו להבחנה בין נתון לנתון. אין חשיבות לכך שנפריד בין שני נתונים בהקשת תו-רווח אחד או יותר, כי תמיד יופיע בפלט תו-רווח אחד בלבד בין שני נתונים.

לעתים יש צורך להדפיס תו-רווח אחד או יותר לפני נתונים או ביניהם. כיצד נוכל להורות למחשב להדפיס את תו-הרווח, כך שהוא יופיע על המסך כרווח ' ' (Blank)?

אנו נעזרים לשם כך בתו הלוכסן היורד '\' (Back Slash), אשר מתקבל על-ידי הקשת המקש המתאים. אחרי הלוכסן, מימין, משאירים את תווי הרווח הדרושים. ב-APPLE LOGO אפשר להשתמש בשני מקשים - מקש <CTRL> ובעודו לחוץ יש ללחוץ על מקש האות **'Q'**.

כל תו שיוקש אחרי הלוכסן, בכלל זה גם תו-הרווח, יתקבל על-ידי המחשב כנתון, ולא יפוענח על-פי המשמעות שלו בלוגו.

<u>הערה</u>: לצורך ההסבר בדוגמאות אשר נביא בספר, נשתמש באות 'b' כדי לציין תו-רווח ' '.

אם נכתוב <--- ?PR "\bPEACE

המחשב ידפיס את המילה PEACE, בהשאירו רווח אחד בתחילת השורה:

bPEACE

?

אם נרצה להדפיס את המילה במרחק של חמשה רווחים מתחילת השורה,

נכתוב זאת כך <--- ?PR "\b\b\b\b\bPEACE

נקבל (5 תווי רווח משמאל!) bbbbbPEACE

?

<u>שים לב</u>- כדי להדפיס תו-רווח דרוש לוכסן. תו-רווח שלא נמצא אחרי לוכסן מתקבל על-ידי המחשב על-פי המשמעות שלו בלוגו, כלומר, הוא יורה על סוף נתון.

לפניך עוד דוגמה ונסה להבין כיצד היא פועלת:

?(PR "\bBLANK1 "\bBLANK2 "\b "BLANK3 "\b "\bBLANK4)

נקבל: bBLANK1bbBLANK2bbbBLANK3bbbbBLANK4

שים לב כמה רווחים יש לפני כל נתון. התוכל לומר כיצד זה קורה? לנוחותך בלבד סומן כל רווח בפלט באמצעות האות 'b', אשר אינה מהווה חלק מהתשובה.

ניקוי מסך

ניתן להבחין, שכאשר מסיימים לרשום בשורה האחרונה, נעלמת השורה הראשונה מן המסך כתוצאה מגלגול השורות כלפי מעלה. כך מקבלים שורה חדשה בשורה האחרונה שהתפנתה. גלגול המסך מאפשר הוספת שורה בתחתית המסך להמשך הכתיבה. אך מה נעשה אם נרצה למחוק את כל המסך על מנת להתחיל מחדש בשורה הראשונה שלו?

ההוראה CLEARTEXT

כאשר רושמים הוראה זו מקבלים מסך נקי, כשבפינה השמאלית העליונה מופיע התו '?', ולידו - הסמן המהבהב.
ב-IBM LOGO וב-APPLE LOGO II ניתן להשתמש גם בקיצור **CT**.

לדוגמה, נקיש ---> ?CLEARTEXT
ונקבל בראש המסך משמאל: ?

פעולות החשבון

סימני פעולות החשבון בלוגו הם: חיבור '+'
חיסור '-'
כפל '*'
חילוק '/'

לחיבור ולחיסור משתמשים בסמלים הרגילים המוכרים לנו. בנוסף לפעולת החיסור, מציין סימן המינוס מספר שלילי, אם הוא מופיע **צמוד** למספר משמאל ללא רווח ביניהם.

אם נכתוב ---> ?PR 7 -4
המחשב ידפיס: 7
אך יעיר את ההערה על כך שאין לו הוראה לגבי 4-:

I DON'T KNOW WHAT TO DO WITH -4

לכפל אין אנו יכולים להשתמש בנקודה '.', כמו למשל '3.4' בשל הבלבול אשר יכול להיגרם בינה לבין הנקודה העשרונית. הדבר נכון גם לגבי ה'איקס' כמו '3X4', שיגרום בקלות לבלבול עם האות 'X'. לכן בלוגו, כמו ברוב שפות המחשב, אנו משתמשים בכוכבית '*' לסימול פעולת הכפל: 3*4.

לחילוק, הסמל המקובל ברוב שפות המחשב הוא לוכסן '/' כמו '12/3', ולא הנקודתיים כמו '12:3', אשר נועד להן תפקיד חשוב אחר. ב-APPLE LOGO II יש להקפיד על תו-רווח משני צידי סימן החילוק. אין לרשום '56/7' אלא '7 / 56'.
יש לזכור שהחלוקה באפס אינה מוגדרת.

אם רוצים לקבוע עדיפות כלשהי לביצוע חלק, או חלקים של ביטוי חשבוני, יש להשתמש בסוגרים. הכללים בשימוש בהם, או בהשמטתם, דומים לאלה הנהוגים באלגברה.

בלוגו, כמו במתמטיקה, הביטוי 3*5+2 פירושו (3*5)+2 והתוצאה היא 17. אם רוצים לכתוב 3*(5+2) אשר התוצאה שלו היא 21, יש להשתמש בסוגריים על מנת להבטיח תרגום נכון של הביטוי.

סדר הביצוע של פעולות החשבון הינו כמקובל באלגברה:
א. פעולות הכפל והחילוק מתבצעות לפני פעולות החיבור והחיסור.
ב. פעולות בעלות אותה עדיפות מתבצעות משמאל לימין.
ג. כאשר יש סוגריים מבצעים תחילה את הפעולות הרשומות בתוכם, לפי כללים א ו-ב לעיל.
ד. אם יש סוגריים במספר רמות (כלומר "סוגריים בתוך סוגריים"), מבצעים תחילה את הביטוי שבסוגריים הפנימיים ביותר.

שים לב, - הביטוי 24/6*2 שונה מן הביטוי 24/(6*2),
- הביטוי 2/a+3 שונה מן הביטוי 2/(a+3), וכדומה.

הוראת הבקרה REPEAT

REPEAT היא פקודה לביצוע חוזר של **רשימת** הוראות **מספר** פעמים. ההוראות שיש לבצע **n** פעמים מוצגות בין שני סוגריים מרובעים. המספר **n**, חייב להיות בעל ערך חיובי. ב-APPLE LOGO ערך זה לא יהיה גדול מ-65535.

ערוך בדיקה וגלה בעצמך מהו הערך המקסימלי בגירסה שבידיך.

המבנה הכללי: REPEAT n [הוראה אחת או יותר]

דוגמה ---> `?REPEAT 3 [PR [WELCOME TO LOGO] PR "]`

הפלט יהיה:

```
WELCOME TO LOGO

WELCOME TO LOGO

WELCOME TO LOGO

?
```

משפט זה יגרום להדפסת הברכה 'WELCOME TO LOGO' 3 פעמים, כאשר אחרי כל הדפסה מופיעה שורת רווח. האם גילית למה יש שורת רווח?

* * * * *

תרגילים.

1. כתוב שורת הוראות למחשב להדפסת הפלט הבא:

```
  A     A     A
 AAA   AAA   AAA
AAAAA AAAAA AAAAA
```

2. כתוב הוראה להדפסת הביטוי החשבוני '=(5+8*7-(2+30/3' והערך שלו, לאחר שחושב על-ידי המחשב.
עם ביצוע ההוראה יודפס הפלט על המסך: 5+8*7-(2+30/3)=49

3. כתוב הוראה למילוי המסך במילה 'COMPUTER', אם ידוע לך שיש 24 שורות פיזיות במסך. נסה זאת גם עם השם שלך.

4. כתוב שורת הוראות להדפסת ריבוע הנוצר על-ידי רצף של האות Z, כך שכל צלע תהיה בנויה מ-10 תווי Z למשל.

* * * * *

ארגון ההדפסה

כבר הזכרנו שהמסך מחולק ל-40 עמודות ול-24 שורות, ובגירסה של IBM LOGO - 25 שורות. העמודה '0' מציינת את העמודה הראשונה בכל שורה. תנועת הסמן היא משמאל לימין עד לעמודה ה-'38'. כזכור, עמודה '39' שהיא האחרונה בכל שורה, מיועדת לציון "המשך" באמצעות סימן קריאה או חץ בהתאם לגירסת לוגו שברשותך. ציון השורות נעשה מלמעלה למטה, כאשר הסיפרה '0' מציינת את השורה הראשונה ו-'24' - את האחרונה בגירסה של IBM LOGO, ובגירסה של APPLE - '23' מציין את השורה האחרונה.

<u>הערה</u>: ב-IBM LOGO וב-APPLE LOGO II יש אפשרות להרחיב את המסך ל-80 עמודות על-ידי השימוש בהוראה SETWIDTH. באמצעות הוראה זו ניתן לקבוע n עמודות בשורה הפיזית.

לדוגמה --> ?SETWIDTH 80
כתוצאה נקבל מסך, שבו נוכל לרשום עד 80 תווים בכל שורה.

ההוראה SETCURSOR

ישנה בלוגו הוראה המאפשרת לקבוע את מקומו של הסמן על המסך כדי לערוך את ההדפסה במקום הרצוי לנו. הוראה זו דורשת קלט בצורת **רשימה**, ובה שני ערכים.

ב-IBM LOGO הערך הראשון מורה על השורה (קואורדינטה Y) והערך השני - על העמודה (הקואורדינטה X).

המבנה הכללי: SETCURSOR [y x]

ב-APPLE LOGO II וב-APPLE LOGO הערך הראשון מצביע על העמודה (קואורדינטה X) והערך השני - על השורה (קואורדינטה Y).

המבנה הכללי: SETCURSOR [x y]

להלן הוראה המדפיסה את הברכה HAPPY BIRTHDAY במרכז המסך:

```
?SETCURSOR [11 13] PR [HAPPY BIRTHDAY]
```

בחישוב פשוט מצאנו שהשורה האמצעית במסך היא השורה '11', ועל התו הראשון להופיע בעמודה '13'. לכן נתנו הוראה למקם את הסמן בנקודה (11,13). כאשר תתבצע הוראת ההדפסה, הופעת התו הראשון של הפלט יהיה במקום שבו נמצא הסמן.

<u>הערה:</u> על המשתמשים בגירסה של APPLE LOGO II או APPLE LOGO להציב את הערכים בסדר הפוך, כלומר תחילה קואורדינטה X ואחר-כך Y ובדוגמה שלנו:

```
?SETCURSOR [13 11]
```

נכתוב כעת הוראה הגורמת להופעת ברכה זו והעלמתה לסירוגין מספר פעמים:

?REPEAT 20 [CLEARTEXT SETCURSOR [11 13] PR [HAPPY BIRTHDAY]]

בשל מהירות הביצוע תופיע הברכה ותעלם בקצב מהיר מדי. כדי להאיט את הקצב של לוגו נשתמש בהוראה WAIT.

הוראת ההשהייה WAIT

זוהי הוראת השהייה, המקבלת קלט של ערך מספרי. ערך זה מורה על משך זמן ההמתנה הנדרש מן המחשב, עד לביצוע ההוראה הבאה.

המבנה הוא: **WAIT n**

רק לאחר שיחלוף הזמן הרשום בקלט ביחידות האופייניות לגירסה שבה פועלת התכנית, ימשיך המחשב בביצוע ההוראה העוקבת אחר ההוראה WAIT.

בדוק ומצא מה צריך להיות ערכו של n על מנת שהמחשב ימתין שנייה שלימה.

ברשימת ההוראות של REPEAT נוסיף הוראת השהייה לפני ההוראה CLEARTEXT ואחריה, וכך נאט את הקצב:-

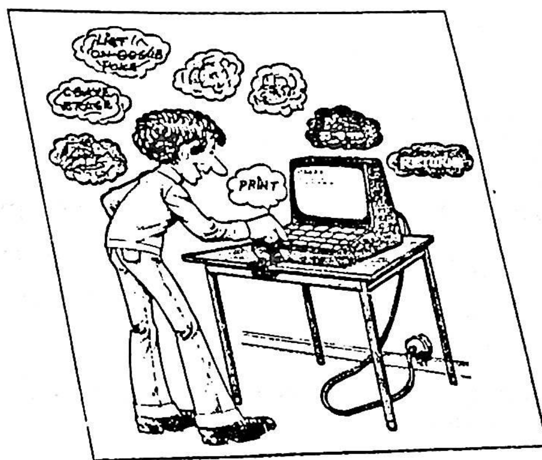
?REPEAT 20 [WAIT 25 CLEARTEXT WAIT 25 SETCURSOR [11 13] !
PR [HAPPY BIRTHDAY]]

* * * * *

תרגילים (המשך)

5. כתוב שורת הוראות היוצרת כרטיס ברכה (מסך ברכה) לשנה החדשה.

6. כתוב הוראה לחישוב מספר הנרות הדרושים לכל ימי חג החנוכה. על ההוראה להציג את כל הביטוי ואת ערכו.



פרק 2

תכנות מודולרי – הליכים

עד עתה כתבנו שורת פקודות למחשב, ולאחריהן הקשנו <RETURN> כדי שהן יחלו להתבצע. אלו הן למעשה פקודות ישירות. נוהגים כך כאשר נמצאים במצב הרגיל של לוגו, והוא מזוהה כאמור, על-ידי סימן השאלה '?' שהוא ה-Prompt של השפה.

במצב זה, עם סיום ביצוע ההוראות, המשפטים שכתבנו לא נשמרים בזיכרון, פרט למשפט האחרון. על כן, אם נרצה שהמחשב יחזור על ביצוע אותן ההוראות, יהיה עלינו לכתבן שוב. מהו, איפוא, הפתרון?

הגדרת הליך

אחת הדרכים לשמירת סידרת הוראות בזיכרון, היא על-ידי הגדרה של **הליך** (Procedure).

ההליך בנוי מ:-

א. **כותרת** המתחילה במילה TO.

ב. **גוף** ההליך המורכב מסידרת הוראות.

ג. המילה **END** לציון סוף ההגדרה.

מילת המפתח היא TO ולידה **שם ההליך**. שם זה הוא לפי בחירתנו כל עוד נשמור על מספר כללים, אך רצוי לבחור בשם בעל משמעות המזכיר את משימת ההליך. שם זה ייהפך בסופו של דבר, למילה של לוגו.

עם לחיצה על <RET>, נעלם סימן השאלה '?' ומופיע במקומו סימן היחס '<' , המציין שנמצאים **במצב של הגדרה**. סימן זה יופיע בתחילת כל שורה חדשה בכל פעם שנסיים שורת הוראות. כל הוראה שתיכתב תהיה חלק מהוראות ההליך, והיא תתבצע **רק** עם הרצתו מאוחר יותר.

שפת לוגו הינה מסוג **פרשן** (אינטרפרטר - Interpreter). כלומר, תרגום משפט ההוראה או ההוראות לשפת מכונה, מתבצע בתום כל שורה. במצב הרגיל, כאשר כותבים פקודות ישירות, הפרשן נכנס לפעולה לאחר כל <RET>. לעומת זאת, בשלבי ההגדרה אין כלל בדיקה, כך שהערות לגבי שגיאות בהליך (אם ישנן כאלו) נקבל עם ביצוע ההוראות בזמן ההרצה.

על מנת לצאת ממצב הגדרה למצב הרגיל של לוגו, יש לכתוב את המילה END בשורה נפרדת ולהקיש <RET>. ברגע זה לוגו תודיע לנו כי **'שם ההליך' מוגדר** ואז יופיע שוב סימן השאלה '?'.

לשם ביצוע ההוראות המרכיבות את ההליך, יש לרשום את שם ההליך ולהקיש <RET> כדי שהמחשב יתחיל בביצוע.

שים לב - כל עוד לא כתבת את המילה END, לא תוכל לפנות ללוגו.

דוגמה להגדרת הליך:
חישוב היקף של מלבן ושטחו, כאשר אורכו הוא 50 ורוחבו הוא 30.

```
?TO MALBEN
>( PR [ THE HEIKEF IS ] 2 * ( 50 + 30 ) )
>( PR [ THE SHETAH IS ] 50 * 30 )
>END
MALBEN IS DEFINED
?
```

כעת נמצא בזיכרון המחשב הליך בשם MALBEN. שם זה מתוסף לאוצר הפקודות של לוגו, באופן שבו נוכל להשתמש בו כמו בכל הוראה אחרת.

נכתוב ---> ?MALBEN

כתוצאה, תתבצענה הפקודות המרכיבות את ההליך ונקבל את הפלט הבא:

THE HEIKEF IS 160
THE SHETAH IS 1500
?

כל מילה שאין גרשיים בתחילתה, מובנת על-ידי לוגו כמשימה שיש לבצע, ולא כרצף של תווים. המחשב קיבל את המילה MALBEN ומכיון שאין פקודה כזו, הוא הבין אותה כשם של הליך, ולא כרצף של תווים. הוא פענח את ההליך וביצע את המשימה המתבקשת מההוראות הכתובות בהליך זה.

אם כתבנו מילה שאינה מוגדרת בלוגו, או טעינו באיות השם וכתבנו MALBIN למשל, האות 'I' במקום האות 'E', היינו מקבלים את ההערה הבאה:

I DON'T KNOW HOW TO MALBIN

גם כאשר נרשום PR COMPUTER, כדי להדפיס את המילה 'COMPUTER' **ונשכח** להקדים אותה בגרשיים, לוגו תתייחס אל COMPUTER כאל הליך. כשלא תמצא אותו נקבל את ההערה:

I DON'T KNOW HOW TO COMPUTER

כל הליך שמגדירים נשמר בזיכרון במהלך העבודה.

מחיקת הליך נעשית באמצעות ההוראה **ERASE**, ובקיצור **ER**. כותבים את ההוראה ואת שם ההליך שברצוננו למחוק, אך יש להקדים את השם בגרשיים.

לדוגמה, ההוראה ERASE "MALBEN, תגרום למחיקת ההליך MALBEN מן הזיכרון. נסו לנחש - מה לוגו תעיר לנו אם נכתוב לאחר מכן

MALBEN? נכון! היא תעיר: I DON'T KNOW HOW TO MALBEN.

כללים לכתיבה של שם הליך

שם ההליך נבחר לפי הכללים הבאים:

א. אין להשתמש במילים הנתונות של לוגו.

אם נכתוב ---> TO REPEAT

נקבל את ההערה הבאה:

REPEAT IS A PRIMITIVE

כל המילים הנתונות בלוגו הן PRIMITIVE, כלומר, מילים קדומות, שהוגדרו בעת יצירת השפה.

הערה: קיימת דרך המאפשרת להשתמש במילים הנתונות של לוגו, אך לא נעסוק בכך בספר זה.

ב. שם יכול להיות מורכב מרצף של אותיות וספרות וכל תו אחר, פרט לתווים המסמלים פעולות אריתמטיות, פעולות יחס, סוגריים עגולים וסוגריים מרובעים. לדוגמה, מילים חוקיות בלוגו: TOTAL, 15%, SHAAR$. אם נרצה להשתמש בשם המורכב משתי מילים, נוכל לכלול בו נקודה לשיפור הקריאות, למשל SHEM.TALMID, או מקף-תחתון-מחבר כמו SHEM_TALMID, אך לא מקף אמצעי המשמש גם לסימן המינוס '-', כמו SHEM-TALMID.

ג. אסור לבחור בשם המורכב כולו מספרות.

אם נכתוב ---> TO 123

נקבל את ההערה הבאה:

TO DOESN'T LIKE 123 AS INPUT

ד. אין לבחור בשם המוגדר כבר בזיכרון.

אם נכתוב שוב <--- TO MALBEN

נקבל את ההערה הבאה:

```
-----------------------------------
MALBEN IS ALREADY DEFINED
-----------------------------------
```

נוכל כמובן לבחור במילה MALBEN1 כי די בתו אחד שונה על-מנת שהשם יהיה שונה.

כתוב כעת הליך הגורם להצגת הפלט הבא:

```
XXXXX   XX  X X X
    X    X  X X X
X   X    X  X X X
X   X    X  X X X
X   X  XXX  XXXXX
```

ההליך SHANA ידפיס X-ים כדי ליצור את המילה **שנה** בשיטת הגרפיקה למחצה (Semi-Graphics).

הגדרת ההליך SHANA

```
?TO SHANA
>PR [\bXXXXX \b\bXX \bX X X]
>PR [\b\b\b\b X \b\b\bX \bX X X]
>REPEAT 2 [PR [\bX \b X \b\b\bX \bX X X]
>PR [\bX \b X \bXXX \bXXXXX]
>PR "
>END
?
```

שים לב, שאחת הפקודות תבוצע פעמיים באמצעות הפקודה REPEAT, כדי להדפיס שתי שורות זהות.

נכתוב SHANA ונקבל את שרטוט המילה **שנה** על המסך. באותה שיטה נוכל לכתוב הליך TOVA המשרטט את המילה **טובה**, או הליך Y1989

ליצירת המספר 1989, או כל מספר אחר.

כאמור, יש באפשרותנו להשתמש בכל הליך שהגדרנו, ואשר נמצא בזיכרון, כמו בכל פקודה אחרת בלוגו.

נוכל לכתוב למשל --->   ?SHANA TOVA SHANA Y1989

המחשב **יפנה** אל ההליכים הכתובים בשורת הוראות זו ויבצע אותם בהתאם לנדרש: תחילה הוא יבצע את ההליך SHANA, לאחר מכן את TOVA, ושוב את SHANA ולבסוף - את ההליך Y1989.

פנייה להליך מתוך הליך

נוכל לפנות אל הליך המוגדר כבר בזיכרון, מתוך הגדרה של הליך אחר. כלומר, הליכים יכולים להיות חלק **מההוראות** של אותו הליך.

דוגמה -

```
?TO NEW.YEAR
>REPEAT 20 [SHANA TOVA]
>END
?
```

נסה למצוא מה יבצע ההליך NEW.YEAR.

האלגוריתם

הליך היא תכנית המורכבת מסידרת הוראות הניתנות לביצוע, כדי לקבל את הפלט הרצוי. סידרת הוראות כזו נקראת **אלגוריתם**.

השם **אלגוריתם** (Algorithm) הוא על שמו של המתמטיקן הערבי אל-חואריזמי מן המאה התשיעית לפנה"ס, אשר חיבר ספרי יסוד באלגברה.

בימי הביניים השתמשו במונח זה עבור תהליך ביצוע של פעולות-חשבון פשוטות על סמך שיטת הסימון ההודית-ערבית. היום משמש

המושג "אלגוריתם", כדי לתאר תהליך של חישוב בסמלים לפי חוקים קבועים הניתן לביצוע על-ידי מכונה, אשר מאפשר פתרון של בעיות נתונות.

בתחום המחשבים, האלגוריתם מתאר את התהליך שעל המחשב לבצע על מנת לחשב ולפלוט את התוצאה המבוקשת. האלגוריתם הפשוט הוא בעל מבנה סידרתי, שההוראות בו מבוצעות זו אחר זו לפי סדר הופעתן בתכנית. מבנה אחר פגשנו בלימוד הפקודה REPEAT, המציגה מבנה של לולאה שבו ההוראות חוזרות על עצמן מספר פעמים.

בהגדרת הליך השתמשנו בפקודות הבסיסיות השייכות לשפה, אשר מוגדרות בדרך שתוכלנה להתבצע על-ידי המחשב ללא פירוט נוסף. המחשב מקבל את ההוראה, מפענח ומבצע אותה. מילה לא מוכרת בשפה גורמת להפסקת הביצוע בליווי הערת שגיאה.

ראינו שנוסף על המילים הבסיסיות של השפה, מאפשרת לוגו להגדיר מילים חדשות לביצוע משימה שהגדרנו בעצמנו, כמו למשל SHANA. הליך זה מורכב מסידרת הוראות יסודיות, שאותן 'יודע' המחשב לבצע.

הגדרת הליכים והשימוש בהם כמו בכל פקודה אחרת המוגדרת בלוגו מאפשרת את יישום התכנות המודולרי. תכנית גדולה ניתן לפצל לכמה הליכים או תת-הליכים, כשלכל הליך תפקיד יותר מצומצם. כך הופכת הגדרת התכנית השלימה לפשוטה יותר, מובנת יותר ונוחה לתיקון.

בתכנות מבני (תכנות מודולרי, או Structured Programming) בונים את התכנית השלימה על-ידי השימוש בהליכים שכבר הוגדרו.

* * * * *

תרגילים

1. כתוב הליך TOVA ליצירת הפלט הבא:

```
XXXXX  XXXX   XX  X XXX
    X     X    X  X X X
X   X     X    X  X   X
X   X     X    X  X   X
X   X  XXXXX   X  XXXXX
```

2. הגדר את ההליך SHANA ושמור אותו בזיכרון המחשב. כתוב לאחר מכן את ההליך NEW.YEAR והרץ אותו.

3. כתוב הליך אשר ישרטט באמצעות אותיות מתאימות את המילה LOGO.

מבנה הפלט:

```
L       000   GGG   000
L      0   0 G   G 0   0
L      0   0 G     0   0
L      0   0 G GGG 0   0
L      0   0 G   G 0   0
LLLLL   000   GGG   000
```

4. כתוב הליך TO HANUKIA ליצירת חנוכיה במרכז המסך, בשיטת Semi-Graphics. המנע משימוש בתווים המשמשים סמלים במתמטיקה.

5. כתוב הליך TO GIL המחשב בשבועות ובימים את גילו של ילד בן שבע שנים. תחילה הוא יחשב את הגיל בשבועות (בהנחה שיש 52 שבועות בשנה) ולאחר מכן - בימים (בהנחה שיש 365 יום בשנה). על המסך יופיע הפלט הבא:

```
THE BOY IS  מספר1  WEEKS OLD
THE BOY IS  מספר2  DAYS OLD
```

במקום **'מספר1'** יירשם הערך שמהווה את גיל הילד בשבועות כפי שחושב על-ידי המחשב. במקום **'מספר2'** - יירשם הגיל בימים, אשר חושב אף הוא על-ידי המחשב.

6. לדני הציונים הבאים: 85, 70 ו-90. חשב והדפס את הממוצע שלהם עם הודעה מתאימה, למשל: THE AVERAGE IS מספר.

7. יוסי כעס מאוד כאשר סבו החליט להעניק לו דמי חנוכה באופן הבא:
ביום הראשון נתן לו שקל אחד.
ביום השני נתן לו כפולה של הסכום שקיבל ביום הקודם.
ביום השלישי נתן לו 3 פעמים הסכום של היום שלפניו.
ביום הרביעי נתן לו 4 פעמים הסכום של היום שלפניו.
:
ביום השמיני נתן לו 8 פעמים הסכום של היום השביעי.
בסיומו של חג החנוכה היה יוסי מאושר מאוד.

- כתוב הליך המחשב את הסכום שקיבל יוסי ביום השמיני.
- מהו סכום דמי החנוכה שקיבל יוסי, אם ביום הראשון קיבל שקל אחד בלבד? ומהו הסכום של דמי החנוכה אם קיבל ביום הראשון 4 שקלים?



פרק 3

עריכת הליכים

עד כה הכרנו את מסך העבודה של לוגו. במסך זה אנו נותנים הוראות ולוגו מבצעת, ואם שוגים במתן הוראה כלשהי - לוגו מעירה. במסך זה גם מגדירים הליכים ומריצים אותם. במסך העבודה לוגו מגיבה בהתאם להוראות שהיא מקבלת.

במסך העבודה אין אפשרות לערוך שינויים, או לתקן שגיאות, אלא רק בשורה שבה נמצא הסמן. גם לא ניתן להוסיף הוראה ששכחנו לרשום בהליך. לא פעם היינו צריכים להגדיר את כל ההליך מחדש בשל מגבלה זו.

למזלנו, מלבד מסך העבודה, ישנו מסך אחר המיועד לעריכה בלבד, בו ניתן לכתוב הליכים, לערוך בהם שינויים, לתקן שגיאות ולהוסיף הוראות במקרה הצורך. בנוסף, אין במסך זה מגבלה של מספר תווים לשורת הוראה, כפי שקיים במסך העבודה (128 תווים).

ההוראה ED (EDIT)

הכניסה למסך העריכה נעשית עם ההוראה ED (עורך - Editor). לאחר הקשת <RET> מקבלים מסך ריק, כשהסמן המהבהב נמצא בפינתו השמאלית העליונה. בתחתיתו נמצא שורה מוארת ובה כתוב **LOGO EDITOR**, אך לא נמצא בה את סימן השאלה. נוכל לכתוב הליכים בהתאם לכללים שלמדנו, דהיינו, להתחיל ב-TO את שורת הכותרת ולידה לכתוב את שם ההליך, למשל GIL. בשורות הבאות רושמים סידרת הוראות ולבסוף, בשורה נפרדת, את מילת הסיום END.

אמרנו 'לכתוב', אך לא אמרנו 'להגדיר'. כלומר, המסך משמש כטיוטה בלבד ולכן לוגו **לא תגיב** על שום הוראה שנרשום במסך זה, אפילו לא אחרי שסיימנו ב-END.

היציאה מן העורך אל מסך העבודה נעשית בשתי דרכים:

1. אישור ללוגו לשמור את אשר כתבנו בזיכרון המחשב, ולקבוע שהליך יהיה 'מוגדר'.
   - ב-IBM LOGO הדבר נעשה באמצעות מקש <**Esc**>.
   - ב-APPLE LOGO II עושים זאת בהקשת מקש **'תפוח'** ומקש האות **'A'** בו זמנית.
   - ב-APPLE LOGO - בלחיצה על מקש <**Ctrl**> והאות **'C'** בו-זמנית. ואכן עם המעבר למסך העבודה נקבל את ההודעה: **GIL IS DEFINED**.

2. יציאה מן העורך ללא הגדרה, מבלי להתחשב אם בוצע שינוי, תיקון או כתיבת הליך חדש. כל מה שנכתב ישאר רק בעורך, בשטח העבודה השמור לו בזיכרון המחשב.
   בדרך זו, ההוראות שנכתבו זה עתה לא תהיינה מוגדרות בלוגו, ולא נקבל את ההודעה על הגדרה כלשהי.
   - ב-IBM LOGO נעשה זאת בלחיצה על <**Ctrl**> ועל המקש <**Break**>.
   - ב-APPLE LOGO II - בלחיצה על מקש **'תפוח לבן'** ומקש <**Esc**>.
   - ב-APPLE LOGO - בלחיצה על <**Ctrl**> והאות **'G'**.

קיימות אפשרויות אחדות לכניסה ל-EDITOR:

1. **כניסה עם ED בלבד:**

א) אם טרם השתמשנו בעורך בהרצה זו, נותנים את הפקודה ED ומקבלים את המסך לפי המתואר בעמוד הקודם.

ב) אם השתמשנו כבר בעורך באותה הרצה, נכתוב ED ונקבל את מסך העריכה **האחרון** שהיה לפני היציאה. אין זה חשוב באיזו דרך יצאנו מן העורך, עם הגדרה או ללא הגדרה.

2. **כניסה בצירוף שם הליך:**

ED "שם

?ED "YEAR

לדוגמה:

א) אם YEAR הוא שם הליך חדש שעוד לא הוגדר, נקבל מסך ריק שבפינתו השמאלית רשום TO YEAR, ואז נוכל להמשיך לכתוב אותו.

ב) אם ההליך YEAR כבר מוגדר, נקבל את מסך העריכה ובו מוצג ההליך כפי שהגדרנו אותו בפעם האחרונה לאחר התיקונים והשינויים.

3. **כניסה עם רשימת שמות של הליכים:**

ED [1שם 2שם ... שםn]

דרך זו מאפשרת לעבור לעורך ולקבל יותר מאשר הליך אחד. ההליכים הכתובים ברשימה, אשר הוגדרו קודם לכן, יופיעו במסך העריכה בזה אחר זה.

דבר זה מבהיר, שניתן לכתוב ב-EDITOR מספר הליכים זה אחר זה. אך **זכור**, אין להתחיל כותרת של הליך חדש לפני שסיימת הליך קודם.

כיצד עובדים במסך העריכה?

ישנן פקודות שונות המקלות על העבודה. הן מאפשרות לנוע על פני המסך באמצעות הסמן כדי לערוך תיקונים או שינויים במקום הרצוי. בהמשך נציג את הפקודות לגירסאות לוגו, אשר אנו דנים בהן בספר זה.

פקודות עריכה עבור IBM LOGO

רוב פקודות העריכה מבוצעות על-ידי הקשה על מקש אחד בלבד, אך ישנן כאלו שמבוצעות בצירוף מקש <Ctrl>, או בעזרת <Shift>.

להלן פקודות העריכה, אשר מתאימות גם למסך העבודה וגם למסך העריכה.

{ Del }	מחיקת התו שעליו מוצב הסמן.
{ <-- }	(חץ אחורה) מוחק את התו שמשמאל לסמן.
{ Tab }	העברת הסמן לסוף השורה.
{Shift + Tab}	העברת הסמן לתחילת השורה.
{ --> }	הזזת הסמן תו אחד ימינה (חץ ימני).
{ <-- }	הזזת הסמן תו אחד שמאלה (חץ שמאלי).
{Ctrl + --> }	מחיקה של התווים בשורה, מן הסמן ימינה.
{Ctrl + <-- }	העתקה של השורה האחרונה שנמחקה.
{ F3 }	העתקה של השורה האחרונה שנכתבה על-ידי המשתמש.
{Ctrl + Break}	הפסקה של העבודה המתבצעת על-ידי המחשב.

הפקודות הבאות מתאימות עבור מסך העריכה (EDITOR) בלבד:

{ חץ-מעלה }	העברת הסמן שורה אחת מעלה.
{ חץ-מטה }	העברת הסמן שורה אחת מטה.
{ END }	העברת הסמן לסוף המסך.
{ Home }	העברת הסמן לתחילת המסך.
{ Pg Up }	דפדוף במסכים אחורה (מעלה).
{ Pg Dn }	דפדוף במסכים קדימה (מטה).
{Ctrl + PgUp}	העברת הסמן לתחילת העריכה.
{Ctrl + PgDn}	העברת הסמן לסוף העריכה.
{ Ins }	פתיחה של שורה חדשה בין שתי שורות קיימות.
{ Esc }	יציאה מן העורך עם הגדרה.

{Ctrl + Break} יציאה מן העורך עם הפסקת העבודה ללא הגדרתה.

{ Ret } העברה של חלק השורה שמן הסמן ימינה, לשורה חדשה.

{ <-- } מחיקת תו שמאלי וצירוף של השורה שהסמן מוצב בתחילתה, אל המשך השורה הקודמת. במילים אחרות, ביטול של <Ret> בשורה שלפני שורת הסמן (חץ אחורה).

פקודות עריכה עבור APPLE LOGO II

הפקודות בגירסה זו של לוגו נעשות בחלקן בעזרת מקש <Ctrl> וחלקן - בעזרת מקש 'תפוח'.

הפקודות אשר מתאימות גם למסך העבודה וגם למסך העריכה:

{ CTRL + F } מחיקת התו עליו עומד הסמן.

{ DEL } מחיקת התו שמשמאל לסמן.

{ **תפוח** + > } העברת הסמן לסוף השורה.

{ **תפוח** + < } העברת הסמן לתחילת השורה.

{ --> } הזזת הסמן תו אחד ימינה (חץ ימני).

{ <-- } הזזת הסמן תו אחד שמאלה (חץ שמאלי).

{ CTRL + Y } מחיקה של יתרת השורה, מן הסמן ימינה.

{ CTRL + X } מחיקה של כל השורה.

{ CTRL + R } העתקה של השורה האחרונה, שהוכנסה או שנמחקה.

{**תפוח לבן** + ESC} הפסקה של העבודה המתבצעת על-ידי המחשב.

הפקודות הבאות מתאימות עבור מסך העריכה (EDITOR) בלבד:

{ CTRL + O }	פתיחה של שורה חדשה בין שתי שורות קיימות.
{ **חץ-מעלה** }	העברה של הסמן שורה אחת מעלה.
{ **חץ-מטה** }	העברה של הסמן שורה אחת מטה.
{**חץ מעלה + תפוח שחור**}	דפדוף המסך אחורה (מעלה).
{**חץ מטה + תפוח שחור**}	דפדוף המסך קדימה (מטה).
{ 1 + **תפוח** }	דילוג אל תחילת העריכה (הקטע הראשון).
{ 2 + **תפוח** }	דילוג אל הקטע השני של העריכה.
. . .	
{ 9 + **תפוח** }	דילוג אל הקטע התשיעי של העריכה (אחרון).
{ A + **תפוח לבן** }	יציאה מן העורך עם הגדרה.
{ ESC + **תפוח לבן** }	יציאה מן העורך עם ביטול העריכה.
{ Ret }	העברה של חלק השורה שמן הסמן ימינה, אל שורה חדשה.
{ DEL }	מחיקת תו שמאלי, והעברה של כל השורה שהסמן מוצב בראשה אל המשך השורה הקודמת. במילים אחרות, ביטול של <Ret> בשורה שלפני שורת הסמן.

<u>הערה:</u> במצב עריכה ניתן על-ידי לחיצה על {? + **תפוח**} לקבל על המסך את כל המידע בדבר הוראות העריכה.

פקודות עריכה עבור APPLE LOGO

רוב הפקודות מתבצעות בהקשה על מקש <Ctrl> בצירוף מקש נוסף. במקום <CTRL> נרשום את הסמל '^', ואם נכתוב A^ פירושו, שיש להקיש על <CTRL> ובעודו לחוץ יש להקיש על האות 'A'.

הפקודות אשר מתאימות גם למסך העבודה וגם למסך העריכה:

orward	^F	הזזה של הסמן תו אחד קדימה (ימינה).
	^U	כנ"ל.
	-->	המקש 'חץ ימינה' מבצע פעולה זהה.
ack	^B	הזזה של הסמן תו אחד אחורה (שמאלה).
dvance	^A	העברה של הסמן לתחילת השורה.
nd	^E	העברה של הסמן לסוף השורה.
elete	^D	מחיקת התו עליו עומד הסמן.
	^H	מחיקת התו שלפני הסמן (משמאל לסמן).
	<--	'חץ שמאלה' מבצע פעולה זהה.
eep	^K	מחיקה של יתרת השורה מן הסמן ימינה.
	^Y	א. העתקה של השורה האחרונה שנמחקה, אל המקום שבו הסמן נמצא, ואת שאר השורה מזיז ימינה. ב. העתקה של השורה האחרונה שנרשמה.

הפקודות הבאות מתאימות למסך העריכה בלבד:

ext line	^N	העברה של הסמן שורה אחת מטה.
revious line	^P	העברה של הסמן שורה אחת מעלה.
pen	^O	פתיחה של שורה חדשה בין שתי שורות קיימות.
	^M	העברה של יתרת השורה מהסמן ימינה אל שורה חדשה.
	<ret>	במצב עריכה פועל מקש זה כמו ^M.
	^C	יציאה מן העורך עם הגדרה.
et out	^G	יציאה מן העורך עם הפסקת העבודה ללא הגדרה.
	^V	דפדוף למסך הבא של העריכה.
	ESC V	דפדוף למסך הקודם של העריכה.
	ESC >	הזזת הסמן לסוף העריכה.
	ESC <	הזזת הסמן לתחילת העריכה.

פקודות הפועלות על הליכים

ההוראה POTS {Print Out Titles}

על מנת לדעת מה קיים בזיכרון, יש להשתמש בהוראה המציגה על המסך את הרשימה של שמות כל ההליכים הנמצאים בו. אם למשל, ההליכים SHANA, TOVA ו-Y1989 קיימים בזיכרון,

ונרשום ---> ?POTS

נקבל:

```
TO SHANA
TO TOVA
TO Y1989
```

בדרך זו ניתן לבדוק איזה הליכים מוגדרים בזיכרון וכיצד נכתבים שמותיהם, דבר שימנע מאתנו טעויות כתיב.

ההוראה PO {Print Out}

א. הוראה זו מציגה על המסך את כל הפקודות הכלולות בהליך כלשהו. אם נכתוב ---> ?PO "SHANA
על המסך תופיע הגדרת ההליך מהתחלת הכותרת ועד מילת הסיום END. במצב זה אי אפשר לערוך אותו.

ב. להצגת מספר הגדרות (הליכים) משתמשים במבנה הבא:

PO [שם1 שם2 ... שמת]

כאשר שם1...שמת מציינים שמות של הליכים.

ההו־ראה POPS {Print Out ProcedureS}

הפקודה POPS גורמת להצגה של **כל** ההליכים המוגדרים בזיכרון.

ההוראה ERPS {ERase ProcedureS}

למחיקה של **כל ההליכים** שבזיכרון משתמשים בפקודה ERPS.

ההוראה ERASE (קיצור - ER)

הכרנו את ההוראה שם" ER (ERASE) למחיקת הליך אחד מסוים.
ניתן להשתמש בהוראה זו כדי למחוק מספר הליכים המפורטים ברשימה,
לפי המבנה הבא:

ER [שם1 שם2 ... שמn]

ההוראה ERALL {ERase ALL}

הוראה זו משמשת למחיקת כל מה שנמצא בזיכרון.

פרק 4

השימוש במשתנים

זיכרון המחשב מורכב ממספר רב של תאים שבהם מאוחסנים נתונים, ולכל אחד מהם יש כתובת. כאשר רוצים להשתמש בתא מסוים אין מתייחסים לכתובת האמיתית שלו, אלא נותנים לו שם סמלי. לתא קוראים **משתנה** (Variable) משום שהנתון המאוחסן בו יכול להשתנות מספר פעמים במהלך ביצוע התכנית.

הנתון המאוחסן בתא הוא **הערך** (Value) של המשתנה.

הגדרת המשתנים נעשית בעת הכרזת הכותרת של ההליך על-ידי מתן שמות, אשר יופיעו מיד אחרי שם ההליך. למשתנים שונים יש לתת שמות שונים.

על מנת שלוגו תבחין בין שם של הליך לבין שם של משתנה, יש **להצמיד** לתחילת המשתנה **נקודתיים** (:). כלומר, להקיש נקודתיים ואת השם ללא רווח ביניהם, לדוגמה MONE:, TOTAL: ועוד.

הגדרת משתנים בכותרת הליך

בפרק קודם הגדרנו הליך MALBEN כדי לחשב היקף ושטח של מלבן בעל נתונים קבועים. בכל פעם שהרצנו את התכנית, קיבלנו תמיד את אותן התוצאות. כעת נגדיר הליך MALBEN1, ובמקום נתונים קבועים נגדיר בו **משתנים** (Variables), שבאמצעותם נוכל לחשב היקף ושטח של מלבן בכל גודל שהוא בעזרת אותה התכנית.

הגדרת MALBEN1

```
?TO MALBEN1 :ROCHAV :ORECH
>( PR [ THE HEIKEF IS ] 2 * ( :ROCHAV + :ORECH ) )
>( PR [ THE SHETAH IS ] :ROCHAV * :ORECH )
>END
MALBEN1 IS DEFINED
?
```

כדי לחשב את ההיקף והשטח של מלבן כלשהו באמצעות הליך זה, צריך להזין נתונים. מתן ערכים למשתנים נעשה עם קריאת ההליך לביצוע. אם נרצה למשל, שהחישוב יתבצע למלבן בעל אורך של 85 ורוחב של 63 נרשום זאת כך:

```
?MALBEN1 63 85
```

קבלת הערכים למשתנים נעשית בהתאמה לפי סדר הופעתם: הערך הראשון למשתנה הראשון, הערך השני למשתנה השני וכן הלאה, אם דרוש. בדוגמה זו, 63 הוא הערך למשתנה ROCHAV: ו-85 הוא הערך של המשתנה ORECH:.

והתוצאה תהיה:

```
THE HEIKEF IS 296
THE SHETAH IS 5355
```

שים לב: גם ROCHAV: וגם ORECH: הינם משתנים שיש להם ערך. מטרת התרגיל היא להדפיס את ערך החישוב של השטח וההיקף, ולכן לא תחמנו את הביטוי בתוך סוגריים מרובעים. אם היינו כותבים:

```
PR [THE HEIKEF IS 2 * (:ROCHAV + :ORECH)]
```

המחשב היה מדפיס את כל מה שנמצא בין הסוגריים, בלי לחשב את הערך של הביטוי.

משתנים אלה הם **משתנים חופשיים** (Free Variables), שעם סיום הביצוע ערכיהם נעלמים. השימוש בהם מאפשר לנו לבצע אותם חישובים עבור מלבנים שונים, על-ידי מתן ערכים חדשים למשתנים בכל קריאה בהתאם לנדרש. כך הופכים ההליכים לכלליים יותר.

תפקיד המשתנים מצטמצם בתיאור תבנית החישוב, ולכן אין כל חשיבות לשמות בהם משתמשים לסימון משתנים אלה. ביצוע ההליך לא יהיה שונה אם נרשום ZELA1: במקום ROCHAV: ו-ZELA2: במקום ORECH:. עם זאת, רצוי לתת לערכים השונים שמות בעלי משמעות, לשם נוחות העבודה.

ניתן להשתמש באותו שם גם להגדרה וגם למשתנה, מכיון שלוגו יכולה להבחין ביניהם בשל סימן הנקודתיים ':' הצמוד לשם של משתנה, אך לא נמצא לפני הגדרה.

זכור- לפני שם של משתנה יש לכתוב נקודתיים (:) **בלי רווח.**

לדוגמה נכתוב הליך כללי לחישוב ביטוי חשבוני מן הצורה הבאה:

X + Y * Z

הגדרת TARGIL

```
TO TARGIL :X :Y :Z
(PR :X "+ :Y "* :Z "= :X + :Y * :Z)
END
?TARGIL 4 2 6
4 + 2 * 6 = 16
```

כאשר נכתוב --->

נקבל את הפלט הבא:

<u>שים לב</u>: בחלקו הראשון של משפט PR, הופיעו סימן הכפל '*' וסימן החיבור '+' מלווים בגרשיים, כדי לגרום להדפסתם, יחד עם הערכים של X: Y: ו-Z:. בצד ימין של הביטוי נכתבו הסימנים ללא הגרשיים, וזה גרם למחשב לבצע את הפעולות עצמן, לחשב את הביטוי ולהדפיסו. הערכים של המשתנים יכולים להיות מספריים (נומריים), ויכולים להיות אלפא-נומרים (רצף של תווים).

נגדיר הליך המקבל שתי מילים, שהם שני שמות של תלמידים, ומדפיס פלט מן הצורה הבאה:

שם1 AND שם2 ARE GOOD FRIENDS.

הגדרת **FRIENDS**

```
TO FRIENDS :NAME1 :NAME2
(PR :NAME1 "AND :NAME2  [ARE GOOD FRIENDS.])
END
```

בדוגמה זו הנתונים הם שמות המורכבים מרצף של תווים, ולכן יש להקדים כל אחד מהם בגרשיים בעת הקריאה.

נכתוב ---> `?FRIENDS "RONI "ELY`

ונקבל: `RONI AND ELY ARE GOOD FRIENDS.`

בכל קריאה נוכל לתת שני שמות אחרים, והפלט יהיה בהתאם.

כאשר מפעילים הליך המוגדר עם משתנים, המחשב מצפה לקלוט נתונים כמספר המשתנים המוגדרים לאותו הליך. במקרה שלנו הוגדרו שני משתנים, ולפיכך:-

אם נכתוב ---> `FRIENDS "GALI`

או רק ---> `FRIENDS`

נקבל הודעת התראה כמו:

NOT ENOUGH INPUTS TO FRIENDS

או **FRIENDS NEEDS MORE INPUTS**

יש לחזור ולכתוב את שם ההליך ואת מספר הנתונים כנדרש.

* * * * *

תרגילים

1. כתוב הליך **TO GIL1**, הדומה ל-TO GIL שבשאלה 5 בפרק 2, לשם חישוב הגיל בשבועות ובימים. הפעם יוגדר ההליך עם משתנה G:, כדי לקבל את גיל הילד בשנים.

2. כתוב הליך TO IGUL כללי לחישוב קוטר, שטח והיקף של מעגל, על-פי הרדיוס אשר יינתן כקלט.

3. הגדר הליך TO HELLO המקבל שם ומדפיס את המשפט הבא:

HELLO שם, IT'S NICE TO MEET YOU.

4. הגדר הליך עם שני משתנים, האחד מקבל מילה והשני - מספר. על ההליך להדפיס את ההודעה הבאה:

MY NAME IS שם
I AM מספר YEARS OLD

5. כתוב הליך TO TIUL המקבל 4 נתונים ל-4 משתנים:
א. מרחק הנסיעה - DERECH:
ב. מהירות ממוצעת - SPEED:
ג. צריכת דלק (ק"מ לליטר) - KML:
ד. מחיר לליטר דלק - PRJCE:

על התכנית לחשב ולהדפיס:
- זמן הנסיעה הדרוש.
- מחיר הדלק הדרוש לנסיעה זו.

כתוב הסבר ליד המספרים.

6. כתוב הליך עם 2 משתנים. המשתנה YITRA: מקבל את היתרה של חשבון לקוח בבנק, והמשתנה השני TNUA: מקבל את סכום ההפקדה, או המשיכה של הלקוח.
על ההליך לחשב את היתרה החדשה ולהדפיס את ההודעה הבאה:

YOUR TRANSACTION: סכום הפקדה/משיכה , NEW BALANSE: יתרה

שים לב לכך שסכומי ההפקדה/משיכה והיתרה יכולים להיות עם סימן שלילי.

* * * * *

העברת משתנים מהליך להליך

ראינו, שניתן לפנות להליך מוגדר מתוך הליך אחר. ידוע גם שבעת קריאה להליך המוגדר עם משתנים, יש לדאוג לנתונים, אשר יהיו הערכים למשתנים אלה בהתאמה. הדבר נכון גם כאשר הקריאה באה מתוך הליך אחר.

נכתוב כעת הליך GIL2 הפונה ל-GIL1 שבתרגיל 1, לשם חישוב הגיל בשבועות ובימים עבור שלושה ילדים בני 5, 9 ו-12 שנים.

הגדרת GIL2

```
TO GIL2
GIL1 5 PR "
GIL1 9 PR "
GIL1 12 PR "
END
```

כאשר נרשום --->
נקבל את הפלט הבא:

```
?GIL2
THE CHILD IS 260 WEEKS OLD
THE CHILD IS 1825 DAYS OLD
THE CHILD IS 468 WEEKS OLD
THE CHILD IS 3285 DAYS OLD
THE CHILD IS 624 WEEKS OLD
THE CHILD IS 4380 DAYS OLD
```

GIL2 פונה שלוש פעמים ל-GIL1 עבור כל אחד מן הילדים. בכל פניה הוא מעביר ערך למשתנה :G המוגדר ב-GIL1. בפעם הראשונה הוא מעביר את הנתון 5, בפעם השניה - את הנתון 9, ופעם אחרונה - את הנתון 12. בתום כל ביצוע של GIL1, מדפיס המחשב שורה ריקה (על-ידי " PR).

GIL2 יבצע תמיד את החישובים עבור אותם גילאים. אם נגדיר אותו עם משתנים - נהפוך אותו להליך כללי, המבצע בכל הרצה את

החישובים עבור 3 גילאים כלשהם. נכתוב עתה את ההליך GIL3, אשר מציג גישה זו.

הגדרת GIL3

```
TO GIL3 :X :Y :Z
GIL1 :X PR "
GIL1 :Y PR "
GIL1 :Z PR "
END
```

להרצת התכנית יש לרשום את שם ההליך 'GIL3' בצירוף 3 נתונים, כמספר המשתנים ---> ?GIL3 10 6 14

המשתנים מקבלים נתונים אלה בהתאמה: המשתנה X: מקבל את הערך 10, המשתנה Y: מקבל את הערך 6, והמשתנה Z: - את הערך 14. בקריאה ל-GIL1 מועבר הערך של המשתנה המצוין באותה פניה אל המשתנה G: המוגדר ב-GIL1.

אם כן, אפשר לתת ערכים למשתנים על-ידי נתונים קבועים, אלא גם על-ידי העברתם באמצעות משתנים.

פרמטר פורמלי ופרמטר אקטואלי

המשתנים המוגדרים יחד עם ההליך נקראים **פרמטרים פורמליים** (Formal Parameters), אשר משמשים לתיאור תבנית החישוב. רק כאשר קוראים להליך ונותנים ערכים למשתנים, מתבצע החישוב עצמו.

הערכים הניתנים בזמן הקריאה נקראים **פרמטרים אקטואלים** (Actual Parameters). ראינו שניתן להעביר בתור פרמטר אקטואלי לא רק קבוע, אלא גם ערך של משתנה. מכאן שמשתנה של הליך יכול לתפקד גם כפרמטר פורמלי וגם כבעל פרמטר אקטואלי.

משתנה מקומי ומשתנה לא-מקומי

הליך אחד יכול לקרוא (לפנות) להליך מוגדר אחר, כמו למשל GIL3 קורא ל-GIL1. הליך קורא נחשב בדרך כלל להליך ברמה גבוהה יותר מאשר ההליך הנקרא על-ידו. המשתנים המוגדרים בהליכים מוכרים תמיד במהלך התכנית על-ידי הליכים בעלי רמה הנמוכה מרמת ההליך בו הם מוגדרים, אך לא להיפך. כך נמצא, שהמשתנים X: :Y ו-Z: מוכרים על-ידי GIL1, אבל המשתנה G: השייך ל-GIL1, אינו מוכר על-ידי GIL3. משתנים המוכרים על-ידי **תת-הליכים** אחרים בתכנית הם משתנים **לא-מקומיים** (בדוגמה זו, X: למשל). משתנים של הליך המוכרים רק על-ידי אותו ההליך, אך לא על-ידי הליך אחר כלשהו בתכנית הם משתנים **מקומיים** (המשתנה G: למשל).

הערכים של **משתנה מקומי** (Local) וגם של **משתנה לא-מקומי** (Non Local) נעלמים עם גמר ביצוע ההליך שבו הם מוגדרים.

ההוראה PONS {Print Out NameS}

הוראה זו גורמת להצגת השמות של כל המשתנים שבזיכרון ואת ערכיהם כפי שהם קיימים בעת מתן ההוראה.

נניח שההוראה PONS היא חלק מהוראות ההליך GIL1. בכל פעם כאשר GIL3 יפנה ל-GIL1, נקבל נוסף על חישוב הגיל, גם את שמות המשתנים הנמצאים ברגע זה בזיכרון ואת ערכיהם הנוכחיים. כלומר, ניתן מתוך תת-הליכים להתייחס אל משתנים, אשר מוגדרים בהליכים בעלי רמה גבוהה יותר, ללא צורך בהעברת פרמטרים.

אם רשמנו 'GIL3 10 6 14', נקבל:

ערכו של X: הוא 10
ערכו של Y: הוא 6
וערכו של Z: הוא 14

המצב יהיה זהה בשלוש הקריאות. כלומר, הערכים של X:, Y: ו-Z: אינם משתנים במהלך כל התכנית. המשתנה G: מקבל ערך אחר בכל קריאה, ולכן נקבל בפעם הראשונה שערכו הוא 10, בדומה לערכו של X:, בפעם השניה הוא יהיה 6 כמו ערכו של Y:, ובפעם האחרונה הוא יהיה 14 כמו הערך של Z:.

כאשר ההוראה PONS ניתנת מתוך ההליך בעל הרמה הנמוכה יותר, מוצגים בפלט המשתנים המוגדרים גם בהליכים האחרים. לא כן, אם נרשום את ההוראה PONS ב-GIL3. כאן לא נמצא זכר למשתנה G:.

<u>הערה:</u> לעתים יש צורך להגדיר משתנים לא-מקומיים כדי שניתן יהיה להתייחס אליהם בתוך תת-הליכים ברמה נמוכה יותר. יש להקפיד לא להגדיר משתנים אחרים בעלי שמות הזהים להם, כדי להמנע משגיאות אפשריות.

במצב של משתנים בעלי שמות זהים המוגדרים בהליכים שונים, יתייחס כל הליך לערך של המשתנה השייך לו. תת-ההליכים שלהם יתייחסו למשתנים השייכים להליכים המדורגים ברמה הקרובה יותר לרמתם.

לדוגמה, נתונים שלושה הליכים:

```
TO AAA :K :L : M
PONS
END
```

```
TO BBB :L :M :N
PONS
PR []
BBB :K :L :L
END
```

```
TO CCC :X
PONS
PR []
CCC 7
END
```

כאשר נרשום ---> ?AAA 2 3 4

נקבל מביצוע ההוראה PONS אשר ב-AAA את התוצאות האלו:

```
M IS 4
L IS 3
K IS 2
```

מביצוע PONS אשר ב-BBB נקבל:

N IS 3
M IS 3
L IS 2
K IS 2

כלומר, הליך BBB מתעלם ממשתנים המוגדרים בהליך AAA ששמותיהם זהים לאלה שלו.

מביצוע PONS אשר ב-CCC נקבל:

X IS 7
N IS 3
M IS 3
L IS 2
K IS 2

ההליך CCC מכיר ב-L: ו-M: אשר ב-BBB, אך לא ב-AAA.

כאשר לוגו מבחינה במהלך התכנית בשם של משתנה בלתי מוכר, היא מפסיקה את הביצוע ומעירה לנו על כך.

אם למשל ההוראה 'PR :G' תהיה חלק מהוראות ההליך GIL3, נקבל את ההערה הבאה:

G HAS NO VALUE IN GIL3

שם בלתי מוכר יכול להיות גם כתוצאה מטעות כתיב, אם רשמנו למשל ROHAV: במקום ROCHAV:. לעתים קורה שלוחצים במידה רופפת על מקש מסוים, ולכן הוא לא נקלט על-ידי המחשב. אם נרשום לדוגמה 'PR : ' בתוך ההליך GIL1 במקום 'PR :G', נקבל:

HAS NO VALUE IN GIL1

שים לב: רשמנו משתנה ללא שם (רק נקודתיים) ולכן הובן כמשתנה ששמו מורכב מהמילה הריקה (ללא תו כלשהו).

הערה: אנו עוסקים בספר זה במשתנים מקומיים ולא-מקומיים בלבד. לכן, אם נרשום PONS במצב הרגיל של לוגו בהוראה ישירה, לא יוצג אף משתנה על המסך.

* * * * *

ביטוי כערך של משתנה

משתנים יכולים לקבל ערכים בדרכים שונות וביניהן - באמצעות ביטוי חשבוני. במקרה זה משמש הביטוי החשבוני לשם העברת ערך למשתנה.

נגדיר הליך MALBENIM הפונה ל-MALBEN1 אשר הגדרנו בתחילת הפרק, לשם חישוב שטחו של מלבן אחד והיקפו על-פי אורך :Y ורוחב :X. לאחר מכן נחשב את השטח וההיקף של מלבן שני, אשר אורכו קטן ב-4 יחידות מן הראשון ורוחבו גדול ב-6 מהראשון.

הגדרת MALBENIM

```
TO MALBENIM :X :Y
MALBEN1 :X :Y
PR "
MALBEN1 (:X + 6) (:Y - 4)
PR "
END
```

אם נתונות הצלעות 24 ו-10, נרשום ---> ?MALBENIM 10 24

המשתנה :X מקבל את הערך 10, ו-:Y את הערך 24.

MALBENIM פונה פעמיים ל-MALBEN1, פעם אחת הוא מבצע את החישוב עבור 10 ו-24, כאשר הערך של :X מועבר כפי שהוא למשתנה :ROCHAV

של MALBEN1, והערך של Y: מועבר ל-ORECH:, אף הוא כפי שהוא. בפעם השניה מועבר הערך של הביטוי (X+6:) ל-ROCHAV:, והערך של הביטוי (Y-4:) מועבר ל-ORECH:.

* * * * *

תרגילים (המשך)

7. כתוב הליך TO 3HELLO, המוגדר עם שלושה משתנים המקבלים 3 שמות שונים. על הליך זה לפנות שלוש פעמים ל-HELLO, אשר הוגדר בתרגיל 3, על מנת לקבל את המשפט עבור כל אחד משמות אלה.

8. כתוב הליך TO IGULIM המוגדר עם משתנה אחד R: ומקבל כקלט את גודל הרדיוס של מעגל נתון. על הליך זה לפנות 3 פעמים ל-IGUL (שהתבקשת להגדיר בתרגיל 2), כדי לבצע את החישובים המוגדרים שם עבור R:, עבור R-1: ועבור R+1:.

* * * * *

<u>יש לזכור:</u>

- משתנה כותבים עם נקודתיים צמודים משמאל.
- שם (מילה) כותבים עם גרשיים צמודים משמאל.
- הגדרה (הליך) כותבים ללא כל תוספת.

פרק 5

הוראות שימוש בדיסקט

בפרק הקודם למדנו להשתמש בהוראות המטפלות בהליכים שבזיכרון המחשב: הצגת שמות, הצגת אופן הגדרה על המסך, קריאה לעריכה ומחיקה מן הזיכרון.

במהלך עבודתנו אנו מגדירים הליכים, אשר נרצה להשתמש בהם שוב במועד אחר. אולם כשמסיימים ומכבים את המחשב, כל העבודה נעלמת כלא היתה. בזיכרון המחשב, שהוא הזיכרון הפנימי, נשמרות ההגדרות כל עוד המחשב פועל, אבל כאשר נרצה לשמור דברים חשובים גם אחרי שמסיימים את העבודה, נעשה זאת בדיסקט המשמש **לאחסנה קבועה**.

שמירת העבודה באחסנה קבועה היא חיונית מאוד, ולכן עלינו להכיר את ההוראות המשמשות למטרה זו.

ההוראה SAVE

באמצעות הוראה זו ניתן לשמור בתקליטון (דיסקט - Diskette) את **כל** ההגדרות הנמצאות באותו רגע בזיכרון המחשב, תחת שם **קובץ** שנבחר על ידנו.
השם יכול להכיל תווים כלשהם, אך לוגו לא תקבל שם המכיל תווים המשמשים סמלים במתמטיקה.
בגירסת IBM LOGO שם של קובץ יכול להכיל עד 8 תווים. יש להמנע משימוש בשם המכיל נקודה, כי שלושת התווים הבאים מימין לנקודה יתקבלו כסיומת, במקום הסיומת המזהה את קבצי לוגו (LF). דוגמה לשם קובץ של לוגו: SHMIRA.LF.

אין ההוראה SAVE מתייחסת אל הליך (Procedure) בודד, אלא אל אוסף הליכים, והיא יוצרת **קובץ** (File) אחד המכיל את כולם.
לשם הבהרה נתאר זאת כך:

- שם של קובץ מקביל לשם של ספר.
- שם של הליך מקביל לשם של פרק או סיפור.

קובץ יכול להכיל הגדרה אחת או יותר, כשם שספר יכול להכיל סיפור אחד או סיפורים אחדים.

אם קיימות בזיכרון ההגדרות GIL1, GIL, IGUL, MALBEN ו-YEAR, ונרצה לשמור אותן בתקליטון תחת שם קובץ TARGILIM.
נרשום ---> ?SAVE "TARGILIM
עם הקשת <RET> מתבצעת פעולת השמירה ונדלקת הנורה שבכונן כדי לציין שהוא פועל.

הקובץ TARGILIM יכיל את כל ההגדרות שהיו בזיכרון המחשב בזמן מתן ההוראה. יש גירסאות בהן לוגו מודיעה לנו כמה הליכים נשמרו בפעולה זו:

----

5 PROCEDURES SAVED

לוגו לא מאפשרת שמירת הגדרות תחת שם קובץ שכבר קיים בדיסקט, וזאת מטעמי זהירות. אם ניתן הוראת SAVE לשם קובץ שכבר קיים, נקבל את ההודעה הבאה:

FILE ALREADY EXISTS

ההוראה DIR

הוראה זו, השייכת לגירסה של IBM LOGO, מציגה על המסך את כל שמות **הקבצים** שבדיסקט. הדבר מאפשר לנו לדעת אילו קבצים קיימים בדיסקט ומה הם שמותיהם. כל שם קובץ אשר נכתב בשפת לוגו יופיע

עם הסיומת .LF, שפירושה Logo File.

לדוגמה: TARGILIM.LF

עבור הגירסה של APPLE LOGO II והגירסה של APPLE משתמשים **בהוראה CATALOG**, בגירסה של APPLE LOGO שם הקובץ שנכתב בשפת לוגו יופיע בסיומת **.LOGO**.

ולדוגמה: TARGILIM.LOGO

הצגת השמות בביצוע ההוראה DIR (או CATALOG) קובעת שההגדרות נמצאות בתקליטון, אך היא אינה מציינת אם הן נמצאות בזיכרון המחשב. כדי שתהיינה בזיכרון יש לטעון אותן מן התקליטון.

ההוראה LOAD

ההוראה LOAD ולידה שם קובץ מטעינה את הזיכרון בכל ההגדרות השמורות תחת שם קובץ זה.

כאשר נכתוב ---> ?LOAD "TARGILIM

נדלקת הנורה שבכונן והטעינה מתבצעת. בגמר הפעולה מופיעה שוב סימן השאלה ולידו הסמן המהבהב. כאשר נרשום POTS, יופיעו על המסך שמות ההליכים שהטענו יחד עם ההליכים שהיו קודם בזיכרון, אם היו כאלה.

אם רוצים לבדוק אילו הגדרות מכיל קובץ מסוים, רצוי לבצע טעינה כשהזיכרון נקי.

אם רושמים שם של קובץ שאיננו קיים, או כותבים את השם בשגיאת כתיב, נקבל את ההודעה הבאה:

FILE NOT FOUND

ההוראה ERASEFILE

למחיקת קובץ מן התקליטון רושמים את ההוראה ERASEFILE ולידה - את שם הקובץ שברצוננו למחוק.

אם נכתוב ---> ?ERASEFILE "TARGILIM

נמחק את קובץ TARGILIM. כאשר נרשום לאחר מכן CATALOG (או DIR), לא נמצא את TARGILIM ברשימת שמות הקבצים שבדיסקט.

כאשר שומרים הליכים בקובץ, אפשר לשמור בו גם הליכים שאינם דרושים, או הליכים שכבר קיימים בקובץ אחר, ובכך לגרום לבזבוז מקום בתקליטון. לכן, רצוי לפני פעולת שמירה לבדוק את תוכן הזיכרון (למשל, בעזרת ההוראה POTS), ולמחוק באמצעות ההוראה ER את ההליכים שאינם נחוצים.

אם נרצה להוסיף הליכים לקובץ קיים, או לאחד כמה קבצים, יש לטעון אותם לזיכרון, לצרפם יחד ולבצע שמירה לשם קובץ חדש. אחר כך יש למחוק את הקבצים הישנים מן התקליטון.

לדוגמה: בדיסקט קיים קובץ בשם RECTANGLE ובו שני הליכים MALBEN1 ו-MALBENIM, וקובץ אחר בשם CIRCLE ובו שני הליכים IGUL ו-IGULIM.

על מנת לאחד את שני הקבצים לקובץ אחד, יש לטעון את שניהם לזיכרון המחשב.

נרשום ---> ?LOAD "RECTANGLE

ואחר כך נרשום ---> ?LOAD "CIRCLE

נניח שלפני פעולת הטעינה הגדרנו הליך בשם RIBUA, אשר נמצא בזיכרון. כעת יהיו בזיכרון 5 הליכים: MALBEN1, MALBENIM, IGUL, RIBUA, ו-IGULIM. אם נרצה, נוכל לשמור את כל החמישה תחת שם קובץ חדש למשל:

?SAVE "HANDASA

לאחר פעולת השמירה נמחק את הקבצים הישנים RECTANGLE ו-CIRCLE מן התקליטון.

קיימות דרכים מתוחכמות יותר לפעולה זו, שלא נעסוק בהן כאן.

ההוראה EDITFILE

(הוראה זו אינה קיימת ב-APPLE LOGO)
נרשום את ההוראה ולידה שם-קובץ,
לדוגמה EDITFILE "TARGILIM.LF

בגירסת APPLE LOGO II יש לרשום את שם הקובץ ללא הסיומת (LF.).

בעזרת הוראה זו שולפים מן התקליטון אל מסך העריכה ושטח העבודה את כל ההגדרות השמורות תחת שם הקובץ הנתון. אין מבצעים את ההליכים.

יוצאים מן העורך (עם שינויים או בלעדיהם) בהתאם לגירסת לוגו שבה פועלים, והקובץ חוזר לדיסקט. בגירסה של IBM LOGO נשאר הקובץ בשטח העבודה של ה-EDITOR ולכן, ניתן לרשום ED ולקבלו שוב במסך העריכה. באמצעות מקש <Esc> אפשר לצאת מן העורך, אך הפעם - להגדיר את הקובץ בזיכרון המחשב לשם ביצוע.

פרק 6

משפטי תנאי

הוראות שכתבנו עד כה בהליכים התבצעו לפי סדר כתיבתן. המחשב ביצע הוראה אחר הוראה עד לסיומו של ההליך. כבר פגשנו בלימוד ההוראה REPEAT, שהמחשב מבצע רשימת הוראות **מספר** פעמים, למרות שהן כתובות פעם אחת. כלומר, הוא חוזר לבצע את ההוראות שברשימה פעם אחר פעם, עד מתי? המחשב עורך השוואה בין מספר הפעמים שההוראות התבצעו לבין הערך המספרי 'ת' הנמצא מיד אחרי ההוראה REPEAT, ומפסיק את הלולאה ברגע שקיים שוויון. כך שבהוראה עצמה כלול משפט תנאי, שלפי קיומו או אי-קיומו של התנאי, תלוי המשך ביצוע הלולאה.

את משפטי התנאי ניתן להציג למחשב גם באמצעות ביטויים לוגיים. השימוש במשפטים אלה דרוש אם רוצים:

א. לבצע רשימת הוראות **רק** אם התנאי מתקיים.

ב. לבצע רשימה אחת של הוראות **כאשר התנאי מתקיים** ורשימת הוראות אחרת **אם התנאי איננו מתקיים**.

אחת הדרכים להציג ביטויים לוגיים היא על-ידי השוואת יחסים באמצעות פעולות היחס:

obj1 = obj2	(obj2 **שווה ל** obj1)
obj1 > obj2	(obj2 **גדול מ** obj1)
obj1 < obj2	(obj2 **קטן מ** obj1)

הוראת התנאי IF

את ההוראה IF אפשר להציג בשני אופנים.

א. מבנה אחד של ההוראה:

IF (ביטוי לוגי) [רשימת הוראות 1]

המחשב בודק אם הביטוי הלוגי המופיע במשפט הוא **אמיתי** או **שקרי**, כלומר - 'נכון' או 'לא נכון'.

- אם התשובה היא חיובית, משמע שהתנאי מתקיים, ואז מתבצעת רשימת ההוראות המופיעה מיד אחרי הביטוי הלוגי שבאותו משפט. המשך התכנית תלוי כמובן, בהוראות שברשימה.
- אם התשובה היא שלילית, המשמעות היא שהתנאי לא מתקיים, ואז מתעלם המחשב מרשימת ההוראות, והביצוע עובר להוראה העוקבת הבאה אחרי משפט ה-IF.

ב. מבנה שני של ההוראה:

IF (ביטוי לוגי) [רשימת הוראות 1] [רשימת הוראות 2]

במבנה הזה כלולות בתוך משפט אחד שתי האפשרויות, שרק אחת מהן תתבצע:

- אם התנאי מתקיים תתבצע 'רשימת הוראות 1' בלבד.
- אם התנאי לא מתקיים, תתבצע רק 'רשימת הוראות 2'.

לפניך הליך המקבל שני מספרים a ו-b. התכנית מדפיסה בשורה הראשונה את שני המספרים שהתקבלו ובשורה השניה - אחת משתי ההודעות, בהתאם לערכי המספרים:

נקבל ---> 'A IS GREATER THAN B'

או ---> 'B IS GREATER THAN A'

הליך AB למציאת המספר הגדול

```
TO AB :A :B
(PR [A =] :A [\b\b\b\b] [B =] :B)
IF :A > :B [PR [A IS GREATER THAN B]]
IF :B > :A [PR [B IS GREATER THAN A]]
END
```

כאשר נכתוב --->

```
?AB 53 12
```

נקבל את הפלט הבא:

```
A = 53        B = 12
A IS GREATER THAN B
```

במקרה הזה, התנאי שבמשפט הראשון מתקיים. כלומר, הערך של A גדול יותר מהערך של B, ולכן מתבצעת ההוראה המופיעה במשפט זה.
במשפט השני אין התנאי מתקיים, לכן התעלם המחשב מההוראה והמשיך הלאה להוראה העוקבת.

התדע מה יהיה הפלט כאשר נרשום --->

```
?AB 6 34
```

או אם נרשום --->

```
?AB 9 9
```

<u>הערה</u>: בלוגו יש אפשרות להשתמש באותיות בעברית כדי להציג הודעות למשתמש ולקבל ממנו נתונים. המעוניין בכך יכול לעיין בנספח הדן בכך. למשל, בהליך AB ייכתב המשפט:

```
IF :A > :B [PR [B הערך של A גדול מן הערך של]]
```

הפלט לאחר ביצוע משפט זה:

```
הערך של A גדול מן הערך של B
```

כעת נכתוב הליך המדפיס את ערכו המוחלט של מספר כלשהו.

הליך MUHLAT

```
TO MUHLAT :X
IF :X < 0 [PR -:X] [PR :X]
END
```

תכנית זו מקבלת ערך למשתנה X, ובודקת אם ערך זה הוא שלילי:

- אם X שלילי - היא כופלת אותו בערך 1- ומדפיסה (רשימה 1).
- אם X לא שלילי, כלומר, אם התנאי לא מתקיים - היא מדפיסה אותו כפי שהוא (רשימה 2).

התכנית תקינה מבחינה לוגית, כי היא עונה על הנדרש - הדפסת ערכו המוחלט של מספר נתון.

הבה נראה את התכנית שנית, כאשר תלמיד אחד הגדיר הליך זה בשגיאה לוגית. התלמיד טעה בכך שהוא כתב את ההוראה 'PR :X' בשורה נפרדת עוקבת, ללא שינוי ערך הקלט.

הליך MUHLAT1

```
TO MUHLAT1 :X
IF :X <0 [PR -:X]
PR :X
END
```

כאשר נרשום מספר חיובי ---> MUHLAT1 5

התנאי שבהוראת IF לא יתקיים והמחשב ידפיס: 5

לגבי ערך חיובי נראה שהתכנית נותנת תוצאה נכונה, אך לא כן לגבי ערך שלילי.

אם נרשום ---> MUHLAT -5

המחשב ידפיס: -5

אך גם: 5

ההוראה 'PR :X' שאמורה להתבצע **רק** אם התנאי לא מתקיים, אינה מופיעה במשפט התנאי, אלא בהוראה נפרדת. עם סיום משפט התנאי עובר אליה הביצוע בכל מקרה, ולכן היא תתבצע תמיד.

אם ניתן היה לסיים את התכנית לאחר ביצוע הרשימה שבהוראת IF, היינו נמנעים משגיאה זו.

הוראת הסיום STOP

בדומה ל-END, מסיימת ההוראה STOP את ביצוע ההליך. המילה END נרשמת בשורה האחרונה בהגדרת ההליך בלבד, כדי לציין את סוף ההגדרה. ההוראה STOP, לעומת זאת, יכולה להופיע בכל מקום בהליך כדי לסיים את הביצוע של ההליך במקום בו היא מופיעה.

נכתוב הליך כדוגמת MUHLAT1, ובאמצעות STOP במקום המתאים, נקבל תכנית נכונה.

הליך MUHLAT2

```
TO MUHLAT2 :X
IF :X < 0 [PR -:X STOP]
PR :X
END
```

בתכנית זו, אם ערכו של X קטן מאפס יודפס הערך המוחלט על-ידי הפיכתו ל-(X-), והביצוע **יסתיים מיד**. לעומת זאת, אם הערך גדול או שווה לאפס, כלומר התנאי לא מתקיים, יתעלם המחשב מההוראות המופיעות ברשימה של משפט התנאי ובכלל זה ההוראה STOP. המחשב יעבור להוראה הבאה, המספר המקורי יודפס ורק אז יסתיים ההליך.

<u>שים לב:</u> מאחר שבמשפטי התנאי המילה **IF** פועלת על ביטוי לוגי שערכו **TRUE** או **FALSE**, לוגו תעיר לנו אם נרשום ביטוי שאיננו לוגי. אם לדוגמה נרשום 'IF :X + 3' במקום 'IF :X = 3' נקבל הודעת שגיאה. אם הערך של X: הוא '9', נקבל את ההערה הבאה:

```
--------------------------
12 IS NOT TRUE OR FALSE
--------------------------
```

הוראות תנאי מקוננות

ההוראות שתתבצענה במשפטי התנאי בהתאם לקיום של התנאי, או אי-קיומו, מוצגות בתוך רשימה. רשימת הוראות יכולה לכלול כל הוראה שהיא ועל-כן, גם הוראת תנאי יכולה להיות בתוך רשימה. מכאן שנוכל לכלול הוראת תנאי פנימית בתוך רשימת הוראות של הוראת תנאי חיצונית.

דוגמה:

IF תנאי [IF תנאי [רשימה א'][רשימה ב']] [רשימת הוראות 2]

"רשימת הוראות 1" של הוראת התנאי החיצונית כוללת הוראת תנאי פנימית: [IF תנאי [רשימה א'][רשימה ב']]

הוראת תנאי פנימית יכולה להופיע גם כחלק מ"רשימת הוראות 2". למעשה, כל הוראת תנאי, פנימית או חיצונית, יכולה לכלול הוראת תנאי פנימית נוספת. נציג זאת בדוגמה.

לפניך מוגדר הליך המקבל שלושה מספרים ומדפיס את הגדול מביניהם.

הגדרת TO GREATER

```
TO GREATER :A :B :C
IF :A>:B [IF :A>:C [PR :A][PR :C]] [IF :B>:C [PR :B][PR :C]]
END
```

רשימה 1 ---> `[IF :A>:C [PR :A][PR :C]]`

תתבצע רק אם **כן** :A>:B

רשימה 2 ---> `[IF :B>:C [PR :B][PR :C]]`

תתבצע רק אם **לא** :A>:B

לדוגמה, כאשר נרשום ---> ?GREATER 7 15 3
נקבל: 15

המשתנה A: יקבל את הערך 7, המשתנה B: יקבל את הערך 15 ו-C: יקבל 3..
התנאי 'IF :A>:B' בהוראה החיצונית לא מתקיים, כי 7 לא גדול מ-15.

לכן מתבצעת רשימה 2: [IF :B>:C [PR :B][PR :C]]

התנאי 'IF :B>:C' בהוראה זו **כן** מתקיים, ועל כן הרשימה הראשונה [PR :B] מתבצעת, והפלט הוא הערך של B: כלומר - 15.

רשימת הוראות היא אוסף של הוראות שיש לו התחלה וסוף. אפשר להציג אוסף הוראות זה בתוך תת-הליך, ולפנות אליו במקום המתאים מתוך ההליך הראשי. **תת-הליך** הוא קטע לוגי של ההליך השלם, המוצג בנפרד מסיבות שנציין להלן.

כפי שאנו רואים, לוגו מאפשרת להגדיר הליכים הכוללים אוסף הוראות מסוים ולפנות אליהם מתוך כל הליך אחר. זוהי שפה הבנויה על-פי עקרונות של התכנות המבני (Structured Programming), אשר הוזכרו בפרק 2. משפט המורכב מהוראות תנאי מקוננות הופך למשפט ארוך ומסורבל. לכן, טוב נעשה אם נשתמש בטכניקה של התכנות המבני להגדרת תת-הליכים שייקלו על הבנת התכנית.

נכתוב כעת תכנית לפתרון התרגיל הבא:
נתונים 7 מטבעות שמשקלם זהה, פרט לאחד הכבד יותר מן האחרים. נתונים גם מאזני פלס. יש למצוא את המטבע הכבד על-ידי מינימום של שקילות.

לפתרון התרגיל נסמן את המטבעות במספרים מ-1 עד 7. המספרים מקבילים לאותיות A עד G בהתאמה.

הגדרת MATBEA

```
TO MATBEA :A :B :C :D :E :F :G
IF :A + :B + :C = :D + :E + :F [PR :G STOP]
IF :A + :B + :C > :D + :E + :F [MTB :A :B :C] [MTB :D :E :F]
END
```

עורכים השוואה בין שתי קבוצות, כשבכל אחת 3 מטבעות: קבוצת המטבעות A, B, C וקבוצת המטבעות D, E, F.

אם משקל שתי הקבוצות שווה, תודפס הודעה שהמטבע השביעי כבד יותר והביצוע יעצור:

```
IF :A + :B + :C = :D + :E + :F [PR :G STOP]
```

אחרת - נמשיך לבדוק איזו מבין שתי הקבוצות כבדה יותר:

```
IF :A + :B + :C > :D + :E + :F
```

- אם התנאי מתקיים, משמע שהקבוצה A, B ו-C כבדה יותר, ואז תתבצע רשימה א':

```
[MTB :A :B :C]
```

- ואם התנאי לא מתקיים, תתבצע רשימה ב':

```
[MTB :D :E :F]
```

אנו פונים להליך MTB להמשך הבדיקה של הקבוצה הכבדה, כאשר ערכי המשתנים שבקריאה מועברים בהתאמה למשתנים המוגדרים בהליך MTB.

הגדרת MTB

```
TO MTB :K :L :M
IF :K = :L [PR :M STOP]
IF :K > :L [PR :K] [PR :L]
END
```

הליך MTB מבצע השוואה רק בין שני מטבעות. אם הם שווים - הוא מדפיס את השלישי והביצוע יעצור. אחרת - הוא בודק איזה מהם הוא הכבד ומדפיס את מספרו.

בפניה להליך MATBEA יש לתת 7 נתונים ל-7 המשתנים, 6 משתנים יקבלו את הערך 0, כדי לציין את המטבעות השווים במשקלם. את המטבע הכבד נציין לפי מספרו הסידורי. אם המטבע הכבד הוא מספר 5 למשל, נציין מספר זה עבור הנתון במקום החמישי.

נרשום ---> ?MATBEA 0 0 0 0 5 0 0

הוראת התנאי TEST

TEST היא הוראה לבדיקת ביטויים לוגיים ושמירת ערך התוצאה (TRUE או FALSE) בזיכרון המחשב במשך ביצוע ההליך. בשלב מסוים לאחר מכן יכולה לבוא הוראת תנאי IF, אשר לפניה נרשום IFT או IFF, בכך נבדוק את התוצאה השמורה של בחינת הביטוי הלוגי, ואז:

- אם הביטוי הוא **אמת** תתבצע רשימת ההוראות המופיעה אחרי הוראת התנאי **IFT** (IFTRUE).
- אם הביטוי הוא **שקרי** תתבצע רשימת ההוראות המופיעה אחרי הוראת התנאי **IFF** (IFFALSE).

לשם דוגמה נגדיר מחדש את ההליך GREATER, ונשתמש בהוראה TEST ובהוראות IFT ו-IFF:

הגדרת GREATER1

```
TO GREATER1 :A :B :C
TEST :A > :B
IFT [IF :A > :C [PR :A] [PR :C] ]
IFF [IF :B > :C [PR :B] [PR :C] ]
END
```

הצגת משפטי הוראות תנאי בדרך זו, הינה פשוטה יותר ומובנת על-ידי קורא התכנית.

המשפט בהוראת IF, צריך להיות כולו **בשורת הוראות אחת שלימה**, ללא הקשת <RET> באמצע. כאשר משתמשים בהוראה TEST - משפט הבדיקה

יכול להיות בשורה נפרדת '... TEST', אפשרות 1 בשורה אחת: '...IFT', ואפשרות 2 בשורה אחרת: '...IFF'.

בנוסף, ניתן להשתמש ב-IFT וב-IFF יותר מאשר פעם אחת בהליך, בכל מקום שהוא. ההתייחסות תהיה תמיד אל תוצאות הבחינה של הביטוי שהופיע במשפט TEST האחרון שבוצע.

* * * * *

תרגילים

1. הגדר הליך המקבל מספר למשתנה X, בודק אם הוא חיובי או שלילי, ומדפיס הודעה מתאימה
   - X IS A POSITIVE NUMBER:

   או - X IS A NEGATIVE NUMBER:

2. כתוב הליך המקבל שני מספרים ובודק אם הם שווים או שונים. בהתאם לכך הוא מדפיס הודעה:
   - EQUAL NUMBERS

   או - UNEQUAL NUMBERS

3. כתוב הליך המקבל שני מספרים ומדפיס אותם בסדר עולה.

4. כתוב תכנית המקבלת שלושה מספרים ומדפיסה אותם בסדר עולה. זכור, כי תכנית יכולה להיות מורכבת מהליך אחד או מהליכים אחדים.

5. כתוב הליך המקבל מספר. אם המספר גדול מ-10, תודפס המילה TRUE, ואם לא - תודפס FALSE.

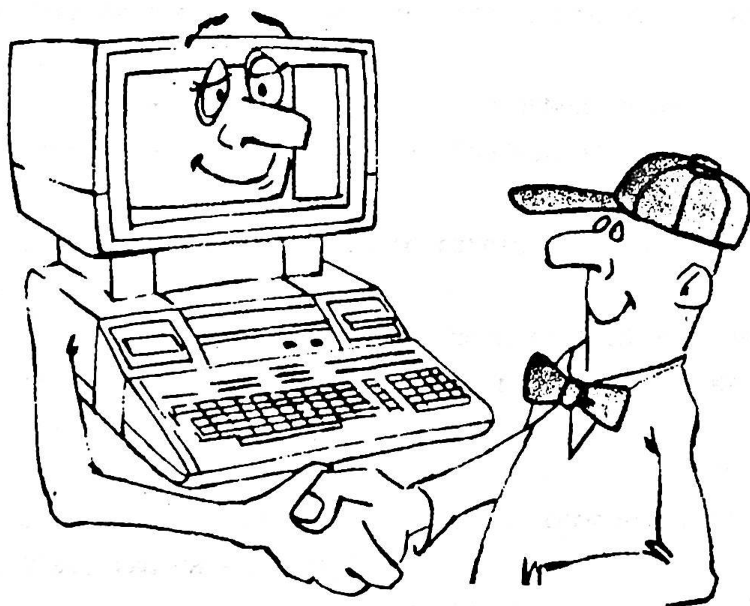
6. כתוב הליך המקבל שני מספרים, ומדפיס את הערך המוחלט של הפרשם.

7. הגדר הליך אשר מקבל מספר למשתנה A: ומספר שני למשתנה B:. הוא בודק אם A: גדול, קטן או שווה ל-B:, ומדפיס הודעה מתאימה.

8. כתוב תכנית לפתרון בעיה הדומה לבעיה MATBEA. עשה זאת עבור קבוצה של 9 מטבעות.

9. כתוב תכנית המקבלת 4 נתונים אשר יכולים לשמש צלעות (לאו דווקא לפי סדר) לבניית מרובע, ובודקת אם אפשר ליצור מרובע שניתן יהיה לחסום בו עיגול.

נתון המשפט: אם הסכום של שתי צלעותיו הנגדיות של מרובע נתון שווה לסכום של שתי הצלעות האחרות, אזי ניתן לחסום בו עיגול. השתמש במשפט זה לצורך בניית התכנית.



פרק 7

רקורסיה

הליך נכתב לביצוע משימה מוגדרת וניתן לפנות אליו על-ידי ציון שמו בצירוף ערכים למשתנים (אם הוא מוגדר עם משתנים). ראינו, שהליך אחד יכול לפנות להליך אחר, והדגמנו זאת בהליכים NEW.YEAR, או 3HELLO.

הליך רקורסיבי

הליך יכול לפנות גם **לעצמו**. כלומר, נכלול את שם ההליך המוגדר כחלק מהוראות ההליך עצמו. הליך כזה נקרא **הליך רקורסיבי** (Recursive Procedure).

נגדיר הליך רקורסיבי TO NUMBERS המדפיס את המספרים הטבעיים.

```
TO NUMBERS :NUM
PR :NUM
NUMBERS :NUM + 1
END
```

הוראה זו הינה פניה להליך עצמו ---> (NUMBERS :NUM + 1)

ביצוע הליך רקורסיבי נעשה **במחזורים**. נדגים זאת בעזרת ההליך שלמדנו.

נבחר במספר '1' כערך התחלתי למשתנה NUM:.

כאשר נפנה ל-NUMBERS ונרשום ---> ?NUMBERS 1

יתבצע המחזור הראשון, ובשל ההוראה 'PR :NUM' נקבל ---> 1

כאן מתבצעת פניה למחזור הבא. פניה זו רשומה כחלק מהוראות ההליך. NUM: מקבל ערך חדש, שהוא הערך העוקב של קודמו, כלומר (NUM+1:), כאילו רשמנו NUMBERS 2. כתוצאה מביצוע המחזור השני

יודפס ---> 2

שוב תתבצע פניה להליך, ומחזור זה יקרא למחזור שלישי.

ערך NUM: יהיה הפעם '3' ויודפס אף הוא על המסך ---> 3

וכן הלאה, מחזור יקרא למחזור, בכל קריאה ערכו של NUM:

גדל ב-1 ובכל ביצוע מודפס ערך חדש ---> 4

5

מתי יהיה על המחשב **לסיים**? - זאת לא אמרנו לו. ולכן, ..

הוא ימשיך הלאה ... הלאה ... ללא הפסקה ... עד אינסוף. ..

28

על מנת להמנע מביצוע אין סופי של מחזורי הפעולה, יש ..

לכלול בתוך ההליך הרקורסיבי **הוראת תנאי לסיום**. 791

..

אם לא נכלול הוראת עצירה בתכנית, או אם נרצה לעצור כל ביצוע שהוא בלוגו, נעשה זאת על-ידי לחיצה על ^C, לחיצה על **Ctrl+Break** או על ESC+**תפוח**, בהתאם לגירסה, ונקבל הודעה מתאימה.

נכתוב עתה את ההליך NUMBERS1 ונכלול בו משפט תנאי לעצירה. נדרוש מהליך זה להדפיס את כל המספרים העוקבים מ-1 עד 100, ובהתאם לכך נקבע את הגבול אשר ישמש תנאי לעצירה.

הגדרת NUMBERS1

```
TO NUMBERS1 :NUM
IF :NUM > 100 [ STOP ]
PR :NUM
NUMBERS1 :NUM + 1
END
```

המספר 1 יהיה הערך ההתחלתי של NUM:, והוא יודפס במחזור הראשון. בכל מחזור NUM: יגדל ב-1, וכשיגיע ל-100 עדיין התנאי להפסקה לא יתקיים, וכך יודפס **גם** המספר 100, **אך לא יותר**.

במקום הביטוי הלוגי '100 > NUM:' יכולנו לרשום '101 = NUM:'. שניהם מתאימים כתנאי הפסקה נכון. אך שים לב - כאשר השתמשנו בסימן היחס '>' (גדול מ-), בחרנו במספר 100 לצורך ההשוואה, אבל לסימן היחס '=' (שווה ל-), התאים המספר 101. השוני בגורם ההשוואה הינו מקור שכיח לשגיאות בכתיבת תכניות ועל-כן רצוי להקפיד.

לסיום ההליך NUMBERS1 השתמשנו **בתנאי עצירה** על מנת להמנע מביצוע אינסופי. עתה נגדיר הליך NUMBERS2 ובו משפט **תנאי לביצוע**, כלומר אם התנאי **לא מתקיים** הביצוע יסתיים.

הגדרת NUMBERS2

```
TO NUMBERS2 :NUM
IF :NUM < 101 [PR :NUM NUMBERS2 :NUM + 1]
END
```

בהליך זה, כל עוד ערכו של NUM: קטן מ-101, המחשב מדפיס ערך זה ומתבצעת פניה נוספת להליך.

נכתוב הליך המדפיס את **סכום** המספרים הזוגיים מ-21 עד 100.

הגדרת SUMZUGI

```
TO SUMZUGI :N :SCHUM
IF :N > 100 [ PR :SCHUM STOP ]
SUMZUGI (:N + 2) (:SCHUM + :N )
END
```

להליך זה שני משתני קלט:

N: לקליטת המספר הראשון,

SCHUM: המציין את הערך ההתחלתי של המשתנה שאליו צוברים את המספרים.

המספר הזוגי הראשון לאחר 21 הוא 22, ולכן הוא ישמש כערך התחלתי ל-N:. ל-SCHUM: ניתן ערך 0 (אפס).

כאשר נרשום ---> SUMZUGI 22 0

נקבל את התוצאה הבאה: 2440

בפניה מתוך ההליך, הערך של N: גדל ב-2, כלומר במחזור השני הוא יהיה 24 במקום 22. אל SCHUM: המשמש כמשתנה צובר, יתוסף ערכו של N: של המחזור הנוכחי (22 + 0). בתחילת המחזור השלישי N: יהיה 26 ו-SCHUM: יהיה הפעם 46 (=22+24), וכך ישתנו הערכים ממחזור למחזור עד לקיום התנאי - 'N: > 100'. רק אז יודפס ערכו של הסכום הכללי (PR :SCHUM), והביצוע יסתיים.

מה היה קורה אילו רשמנו 21 כערך התחלתי ל-N: במקום 22?
הנכון הוא - שהיינו מקבלים את סכום כל המספרים **האי-זוגיים** מ-21 ועד 100. דבר שמביא אותנו להסיק שיש לבחור בקלט הנכון בהתאם להנחיות הבעיה.

עוד דבר שיש להעיר עליו הוא, שאם היינו משתמשים במשפט התנאי בביטוי הלוגי 'N: = 101', הביצוע היה הופך לאינסופי עבור 22 כערך התחלתי. אותו הדבר היה קורה לגבי הביטוי 'N: = 100' עבור חישוב סכום המספרים האי-זוגיים מ-21 ועד 100. התדע מדוע?
סיוע: נסה לחבר כפולות של 2 אל הערך ההתחלתי שקבעת.

בהליך רקורסיבי מתבצעות ההוראות שוב ושוב, כאשר הערכים של המשתנים עליהם מתבצע החישוב **יכולים** להיות שונים ממחזור למחזור, בהתאם לדרישות התרגיל. מה **שכן צריך** להשתנות, זהו ערכו של הביטוי הלוגי שבו תלוי התנאי לעצירה או התנאי לביצוע, ולכן יש להיות זהירים בקביעת התנאי הזה.

* * * * *

תרגילים

1. הגדר מחדש את SUMZUGI, אך הפעם השתמש **בתנאי לביצוע**. כל עוד התנאי מתקיים, הביצוע נמשך (ללא הוראת STOP).

2. כתוב הליך להדפסת המספרים מ-1 עד 50 המסתיימים ב-6.

3. כתוב הליך המדפיס את המספרים מ-1 עד 100 המתחלקים ב-7.

4. כתוב הליך המדפיס **בסדר יורד** מ-200 ל-100 את המספרים המתחלקים ב-8.

5. כתוב הליך להדפסת **סכום** המספרים מ-1 עד 30 המתחלקים ב-3.

3 + 6 + 9 + ... + 30

6. כתוב הליך רקורסיבי להדפסת **מכפלת** המספרים מ-1 עד 10.

1 * 2 * 3 * ... * 10

7. הגדר הליך להדפסת תרגילי כפל של X: במספרים שמ-1 עד 10. אם הערך של X: הוא 6, נקבל:

6 * 1 = 6

6 * 2 = 12

:

6 * 10 = 60

* * * * *

בהוראת REPEAT מתבצעות ההוראות שברשימה **n** פעמים. תנאי ההפסקה של הביצוע החוזר נבדק על-ידי פעולת פיקוח פנימית של לוגו באמצעות מונה. הוא סופר כמה פעמים התבצעה רשימת ההוראות, ומשווה בין מונה זה לבין הערך של **n**. גם שינוי ערכו של המונה וגם הבדיקה כלולים בתוך ההוראה עצמה.

להלן הליך המדפיס N: פעמים את המשפט 'I AM A SUPERMAN', פעם בהליך רקורסיבי ופעם עם ההוראה REPEAT.

הגדרת REC

```
TO REC :N :MONE
IF :MONE > :N [STOP ]
PR [I AM A SUPERMAN ]
REC :MONE + 1 : N
END
```

הגדרת REPT

```
TO REPT :N
REPEAT :N [ PR [I AM A SUPERMAN ]]
END
```

אם ניתן למשתנה N: ערך זהה בשני ההליכים, ולמשתנה MONE: ניתן את הערך 1, הפלט שיתקבל מביצוע ההליך REC יהיה זהה לפלט של REPT. ב-REC המשתנה MONE: מיועד לספור כמה פעמים מתבצעת הרקורסיה. לכן, אם מספר הפעמים הנדרש ידוע (או ניתן לחישוב), וביצוע ההוראות החוזר על עצמו הוא ללא ערכים משתנים, ניתן יהיה להשתמש בהוראה REPEAT.

הדפסה רקורסיבית

נלמד עתה דרכים לנצל את העוצמה של הרקורסיה בפקודות הדפסה שונות.

ההוראה TYPE

בהוראה PRINT, עובר הסמן בתום ההדפסה לתחילת שורה חדשה. כאשר הרצנו את NUMBERS, הודפסו המספרים זה תחת זה. כל הוראת PR קיבלה ערך חדש והדפיסה אותו בשורה חדשה. שונה הדבר בהוראה TYPE, בה לא יעבור הסמן לשורה חדשה, אלא ישאר אחרי התו האחרון שהודפס. שם יודפס הנתון הבא (אם יש כזה).

מבנה כללי: **TYPE דבר**

דוגמה --->

```
TO HAPPY
TYPE "HAPPY
TYPE "NEW TYPE "YEAR
END
```

כאשר נרשום ---> ?HAPPY

נקבל: HAPPYNEWYEAR?

האות N של NEW הופיעה מיד אחרי האות Y של HAPPY, וכך גם לגבי המילה YEAR הצמודה למילה NEW. ובסיום ה-TYPE האחרון הופיע התו

(?) של LOGO באותה שורה, בצמוד לאות R של YEAR. המילים הודפסו ללא רווחים ביניהן.
על מנת לקבל רווח בין נתון לנתון עלינו לבקש הדפסת רווח במקום הדרוש, באמצעות הלוכסן ההפוך (\).

דוגמה ---> ?TYPE "HAPPY\b TYPE "NEW TYPE "\bYEAR TYPE "\b
נקבל: HAPPY NEW YEAR ?

שים לב: התווית (?) הופיעה אחרי תו רווח, שהוא התו האחרון שהודפס (תו הרווח הוא התו של הוראת TYPE האחרונה).

נוכל להשתמש בהוראה TYPE להדפסה של **מספר דברים** בהוראה אחת, אם נתחום את כל המשפט **בסוגריים עגולים**:

?(**TYPE** "HAPPY\b "NEW "\bYEAR "\b)

הוראת TYPE פועלת גם על **רשימה**. בהדפסת רשימה תמיד יופיע רווח בין הנתונים.

דוגמה ---> ?TYPE [HAPPY NEW YEAR]
נקבל: HAPPY NEW YEAR?

גם כאן, התווית (?) של לוגו צמודה לתו האחרון שנכתב.

הדפסת רצף של תווים המשמשים סמלים במתמטיקה

כאשר התבקשת לכתוב הליך היוצר צורות בשיטת ה-Semi-Graphics, יעצנו לך לא להשתמש בתווים השייכים למתמטיקה. הסיבה לכך היא, שלוגו מקבלת כל אחד מן התווים האלה כנתון אחד ויחיד, ולכן לא נוכל לרשום הוראה להדפסת רצף של כוכביות, למשל.

אם נכתוב למשל ---> ?PR [*****]
נקבל: * * * * *

גם אם נרשום ---> ?TYPE [*****]

נקבל: * * * * *?

בין כל שתי כוכביות הופיע תו רווח. הדבר נכון גם לגבי שאר הפעולות החשבוניות, פעולות היחס והסוגריים העגולים.

בלוגו לא נוכל לכתוב: PR "***, וגם לא: TYPE "***. בכל מקרה נקבל הערה, שכן לוגו מוכנה לקבל רק נתון אחד, ז"א כוכבית אחת המלווה בגרשיים, אך לא יותר.

נוכל לנצל את התכונה של TYPE באופן הנכון כדי לקבל את שביקשנו.

אם נכתוב ---> ?(TYPE "* "* "* "*)

או אם נרשום ---> ?REPEAT 4 [TYPE "*]

נקבל 4 כוכבים צמודים: ****?

אחרי כל הדפסה ב-TYPE יעבור הסמן לעמודה הבאה, שם הוא ידפיס את הנתון התורן.

עוד אפשרות להדפסת רצף של סימנים היא על-ידי השימוש בלוכסן, כפי שעשינו בהדפסת רווחים.

לדוגמה ---> ?PR "****

או ---> ?PR [****]

בשיעור הראשון רשמנו הוראה להדפסת הברכה 'HAPPY BIRTHDAY' במרכז המסך, מחקנו והצגנו שוב לסירוגין 20 פעם:

?REPEAT 20 [WAIT 25 CLEARTEXT WAIT 25 SETC... ... ..AY]]

נוכל לעצב את הברכה בצורה נאה יותר, אם נמסגר אותה בכוכביות למשל, או כל תו אחר. לדוגמה:

```
 *****************
*                 *
*  HAPPY BIRTHDAY *
*                 *
 *****************
```

אבל, כאשר נשתמש ב-CLEARTEXT נמחק את כל המסך וכך, גם המסגרת תופיע ותעלם יחד עם הברכה. אבל, אנו רוצים שהברכה בלבד תוצג ותעלם לסירוגין.

כאשר מציבים את הסמן בנקודה (y,x), שבה כבר מופיע תו כלשהו, ומבקשים הדפסת תו אחר, הפלט החדש יעלה על הישן. כלומר, אם נבקש הדפסת תו-רווח במקום שבו קיימת האות 'H' למשל, תעלם אות זו ויופיע במקומה רווח. לכן נוכל לתת הוראה שתמחק את רצף התווים היוצרים את המשפט 'HAPPY BIRTHDAY' בלבד והמסגרת תשאר ללא שינוי.

נרשום שוב את משפט ההוראה REPEAT, עם השינויים המתאימים:

```
?REPEAT 20 [WAIT 25 SETCURSOR [11 13] REPEAT 14 [TYPE "\b] !
          WAIT 25 SETCURSOR [11 13] TYPE [HAPPY BIRTHDAY]]
```

במשפט שלפנינו ההוראות המודגשות הינן במקום ההוראה **CLEARTEXT** שבדוגמה הקודמת.

ההוראה REPEAT 14 מציינת שיש לכתוב 14 תווי-רווח, החל מהמקום המצויין על פני המסך. שים לב לכך שההוראה ארוכה מאורך שורה.

ב-APPLE LOGO, סימן הקריאה בסוף השורה מציין ששורת ההוראות לא מסתיימת כאן, וההמשך מופיע בשורה הבאה. ב-IBM מציינים זאת על-ידי חץ-ימינה (<--).

יש להבחין בין אופן הביצוע של ההוראה PRINT, לבין אופן הביצוע של ההוראה TYPE. ב-TYPE הפעולה מתבצעת על כל עמודה (או תו), ובתום ההדפסה נמצא הסמן בעמודה הבאה לאחר התו האחרון שהודפס. לעומת זאת, ב-PR הפעולה מתבצעת על שורה שלימה. על-כן, אם הפלט המודפס באמצעות ההוראה PR איננו ממלא את מספר התווים שבשורת המסך, מתמלאים התווים הנותרים של השורה ברווחים.

השינויים שעשינו ברשימת ההוראות של REPEAT מבצעים את אשר בקשנו, אבל הם הפכו את המשפט לארוך מאד. דבר זה לא נאה לעין ולא נוח לקריאה. לכן נגדיר הליך HPY, ונכתוב בו את כל ההוראות הכלולות ברשימה של REPEAT במבנה נוח וגם פשוט יותר.

הגדרת HPY

```
TO HPY
SETCURSOR [11 13]
REPEAT 14 [TYPE  "\b ]
WAIT 25
SETCURSOR [11 13]
TYPE [HAPPY BIRTHDAY]
WAIT 25
END
```

הליך זה מציב את הסמן בעמודה 13 אשר בשורה 11, מדפיס 14 תווי-רוות בזה אחר זה, ממתין קמעה, מציב שוב את הסמן בנקודה (11,13), מדפיס את הברכה, וממתין כאן שוב.

הדפסת הברכה ומחיקתה מתבצעות פעם אחת בהליך זה. אם נרצה שהביצוע יחזור 20 פעם,

נוכל לרשום ---> ‎?REPEAT 20 [HPY]

כעת נוכל לכתוב הליך היוצר מסגרת במרכז המסך הבנויה מכוכביות, ובתוכה תופיע הברכה ותעלם מבלי שהדבר ישפיע על המסגרת.

הגדרת BIRTHDAY1

```
TO BIRTHDAY1 :N
CLEARTEXT
SETCURSOR [9 11]
REPEAT 18 [TYPE "*]
SETCURSOR [10 10] TYPE "*
SETCURSOR [10 29] PR "*
SETCURSOR [11 10] TYPE "*
SETCURSOR [11 29] TYPE "*
SETCURSOR [12 10] TYPE "*
SETCURSOR [12 29] TYPE "*
SETCURSOR [13 11]
REPEAT 18 [TYPE "*]
REPEAT :N [HPY]
END
```

בהליך זה מנקים את המסך, מדפיסים את המסגרת ומבצעים N: פעמים את ההליך HPY.

<u>הערה:</u> משתמשי הגירסה של APPLE LOGO II או APPLE LOGO, צריכים להפוך את סדר הנתונים המציינים את הקואורדינטות של הנקודות שעל המסך. תחילה יש לרשום את ערך העמודה ואחר כך את ערך השורה, כלומר (y,x) במקום (x,y).

הפעלת הליך רקורסיבי מתוך הליך רקורסיבי אחר

נגדיר הליך רקורסיבי המדפיס סידרה כלשהי של מספרים עוקבים מ-n ועד m, כשרווח אחד מפריד בין מספר למספר.

הגדרת SIDRA

```
TO SIDRA :N :M
IF :N > :M [ STOP ]
( TYPE :N "\b )
SIDRA :N+1 :M
END
```

אם נכתוב <--- `?SIDRA 7 20`

נקבל: `7 8 9 10 11 12 13 14 15 16 17 18 19 20`

המשתנה N: קיבל את הערך 7 כערך התחלתי, ובכל מחזור הוא גדל באחד. המשתנה M: קיבל את ערך הגבול המהווה את המספר האחרון של הסידרה. בכל מחזור נעשתה בדיקה אם M: < N:. כל עוד N: איננו גדול מ-M:, המחשב מדפיס את ערכו של N: ותו רווח אחד, וממשיך למחזור שלאחריו. כאשר תנאי זה מתקיים מסתיים הביצוע.

נוכל לכתוב הליך STEPS הפונה ל-SIDRA להדפסת המספרים מ-1 עד 9,

כך שבשורה הראשונה יודפס המספר ---> 1

בשורה השניה יודפסו שני המספרים מ-1 עד 2 ---> 1 2

בשורה השלישית יודפסו המספרים מ-1 עד 3 ---> 1 2 3

:

וכן הלאה עד השורה התשיעית ---> 1 2 3 4 5 6 7 8 9

הגדרת STEPS

```
TO STEPS
SIDRA 1 1   PR "
SIDRA 1 2   PR "
SIDRA 1 3   PR "
SIDRA 1 4   PR "
SIDRA 1 5   PR "
SIDRA 1 6   PR "
SIDRA 1 7   PR "
SIDRA 1 8   PR "
SIDRA 1 9   PR "
END
```

ההליך STEPS פונה ל-SIDRA 9 פעמים. בכל קריאה הוא קובע ל-N: את הערך 1, ואילו ל-M: הוא קובע ערך גבולי הגדל משורה לשורה ב-1. בקריאה הראשונה ערך הגבול יהיה 1, בקריאה השניה הוא יהיה 2, וכן הלאה, עד שבקריאה התשיעית M: יקבל ערך 9. אחרי כל קריאה ל-SIDRA ניתנה ההוראה '" PR', כדי שהסידרה הבאה תודפס בשורה חדשה.

הליך זה הינו פשוט, כי לכל פניה יש הוראה נפרדת. ההוראות מתבצעות לפי סדר זו אחר זו, וההליך מסתיים על-ידי ההוראה END. שיטה זו טובה כאשר מספר הצעדים קטן. אך אם נרצה לבצע את הפעולה 500 פעם למשל, האם נצטרך לכתוב 500 שורות של הוראות?

בגלל המבנה הסדיר של ההליך STEPS, ניתן להפוך אותו להליך רקורסיבי, כפי שמובא להלן:

הגדרת STEPS1

```
TO STEPS1 :X
IF :X > 9 [STOP ]
SIDRA 1 :X
PR "
STEPS1 :X+1
END
```

כדי לקבל תוצאה כמו של הליך STEPS המקורי,

נכתוב ---> ?STEPS1 1

למשתנה X: אשר משמש להעברת ערך הגבול ניתן ערך התחלתי 1. מספר השורות שיש להדפיס הוא 9, ולכן התנאי לעצירה הוא כאשר X>9:. בכל מחזור פונה ההליך STEPS1 ל-SIDRA, וקובע ל-N: את הערך 1, ואילו ל-M: מועבר ערך המשתנה X: שהוא כאמור, הערך הגבולי לכל שורה.

במחזור הראשון M: מקבל 1, ולכן מודפס 1 ותו לא. אחרי סיום הביצוע של SIDRA, חוזר המחשב להוראה הבאה לאחר הקריאה '" PR',

הוראה זו מעבירה את הסמן לתחילת שורה חדשה, בה מתבצעת ההדפסה של המחזור הבא (STEPS1 :X+1). בכל מחזור X: גדל ב-1, וכך בכל קריאה מקבל M: ערך גדול ב-1 מאשר בקריאה הקודמת.

* * * * *

תרגילים (המשך)

8. כתוב הליך TAB להדפסת n תווי רווח, לכל ערך שהוא n.

לאחר מכן, השתמש בהליך זה כחלק של הוראה המדפיסה שני שמות, למשל POPEY ו-OLIVE, כש-n תווי רווח מפרידים ביניהם.

לדוגמה, אם נכתוב ---> ?TYPE "POPEY TAB 7 PR "OLIVE
נקבל: POPEYbbbbbbbOLIVE

ההליך TAB קיבל את הערך 7 כקלט, ולכן קיבלנו 7 תווי רווח בין שני השמות (התו b מסמל תו רווח).

9. הגדר הליך המקבל שם כלשהו, ומדפיס אותו על פני כל המסך.

10. הגדר הליך HEZKA המקבל שני מספרים טבעיים, אחד ל-A: והשני ל-X:, מחשב ומדפיס את A: בחזקת X:. כתוב הודעה אשר תציין את שני מספרי הקלט ואת התוצאה.

11. כתוב הליך 'TO DOLLARUP :NUM' להדפסת הפלט הבא:

בשורה הראשונה יופיע סימן דולר **אחד** ---> $
בשורה השניה יופיעו **שני** סימני דולר ---> $$
:
בשורה העשירית יופיעו **עשרה** סימני דולר ---> $$$$$$$$$$

משפט העצירה יהיה 'IF :NUM>10 [STOP]'.

12. כתוב הליך 'TO DOLLARDOWN :MIS' אשר ידפיס

בשורה הראשונה 10 סימני דולר <--- $$$$$$$$$$

ובכל שורה יודפס סימן '$' אחד פחות <--- $$$$$$$$$

:

עד שבשורה העשירית יודפס דולר אחד בלבד <--- $

הערך ההתחלתי של MIS: הוא 10. בכל מחזור יופחת ממנו אחד ועל-כן הוראת העצירה תהיה 'IF :MIS<1 [STOP]'.

13. כתוב הליך להדפסת הפלט הבא:

בשורה הראשונה יודפס המספר 1 **פעם אחת** <--- 1

בשורה השניה - המספר 2 **פעמיים** <--- 22

בשורה השלישית - המספר 3 **שלוש פעמים** <--- 333

:

ובשורה התשיעית - המספר 9 **תשע פעמים** <--- 999999999

14. הגדר הליך להדפסת הפלט הבא:

בשורה הראשונה יודפס המספר 1 אחרי רווח אחד <--- b1

בשורה השניה יודפס המספר 2 אחרי שני רווחים <--- bb2

בשורה השלישית - המספר 3 אחרי שלושה רווחים <--- bbb3

: :

ובשורה התשיעית - המספר 9 אחרי 9 רווחים <--- bbbbbbbbb9

<u>זכור</u>: התו b מסמל תו רווח.

15. הגדר תכנית MESHUP היוצרת משולש כוכביות כפי שמוצג כאן:

```
      *
     ***
    *****
   *******
  *********
 ***********
*************
```

16. הגדר הליך MESHDOWN היוצר משולש כמו בתרגיל 13, אבל הוא יהיה מוצב הפוך: הבסיס למעלה והקודקוד מופנה כלפי מטה.

17. שנה את משפט העצירה שבתרגיל 14, כך שבתום ביצוע MESHUP, תתבצע פניה להליך MESHDOWN ורק לאחר מכן הביצוע יעצור. הצורה שתתקבל במקרה זה תהיה מעוין של כוכביות.

18. כתוב תכנית עם משתנה קלט N: המקבל ערך טבעי ומדפיסה את הפלט כמובא להלן. במקום N: יודפס הערך של המשתנה.

```
1 2 3 4 ... :N
2 3 4 ... :N
3 4 ... :N
:  :N
:N
```

19. כתוב תכנית להדפסת לוח הכפל של 10 על 10.

הפלט יהיה:

```
 1  2  3  4  5  6  7  8  9  10
 2  4        ...          18  20
 .                             .
 .                             .
10           ...             100
```

20. בהנחה שלא קיימת ההוראה WAIT בלוגו, הגדר אותה בעצמך.

21. כתוב תכנית להדמיית שעון דיגיטלי לציון השעה, הדקה והשניה. למשל, אם שעון זה מצביע על השעה 4, 36 דקות ו-7 שניות, נראה זאת על המסך בצורה הבאה: 4:36:07.

<u>הדרכה:</u> תכנית זו תהיה מורכבת מארבעה הליכים: CLOCK, HOUR, MINUTE ו-SECOND.

א. ההליך TO CLOCK

- ינקה את המסך.
- יבנה מסגרת נאה לשעון.
- יקבע מקום להדפסת הנקודתיים המפרידים בין ציון השעות, הדקות והשניות. ויפנה להליך HOUR, עם ערך התחלתי 1.

ב. ההליך TO HOUR :H

- יקבע את המקום שבו תודפס השעה. אם השעה מצויינת על-ידי סיפרה אחת, יש להדפיס רווח במקום סיפרת העשרות.

```
IF :H<10 [(TYPE "\b :H)] [TYPE :H]
```

- לאחר מכן הוא יפנה ל-MINUTE ויתן ערך התחלתי להרצת הדקות. עם תום הביצוע של MINUTE, מתקדמים למחזור הבא לציון השעה הבאה 'HOUR :H+1'. תנאי להפסקה יהיה כאשר השעה היא 12 למשל.

ג. ההליך TO MINUTE :M

- ימקם את הסמן בנקודה בה יודפסו הדקות. אם הדקה היא בת סיפרה אחת, יש להדפיס אפס במקומה של סיפרת העשרות,

```
IF :M<10 [(TYPE "0 :M)] [TYPE :M]
```

- על ההליך לפנות ל-SECOND עם ערך התחלתי '0' להרצת השניות,
- ולאחר מכן להתקדם למחזור הבא 'MINUTE :M+1'. תנאי העצירה יהיה כאשר ':M>59'.

ד. ההליך 'TO SECOND :S' דומה להליך MINUTE, אך בהבדל קטן: הליך SECOND אינו פונה להליך אחר פרט לעצמו 'SECOND :S+1'. בנוסף, יש לכלול בו הוראת השהיה משניה לשניה.

פרק 8

פונקציות נתונות

בלוגו, כמו גם בשפות אחרות, כלולות פונקציות שימושיות שונות. יש הקוראים להן **פונקציות בנויות** (Built-in Functions), כי הן נבנו עם יצירת השפה.

פונקציה, ארגומנט וערך הפונקציה

לפני שנמשיך נסביר את מושג **הפונקציה** (Function).

נתונות שתי קבוצות א' ו-ב', ולכל איבר בקבוצה א' מותאם איבר **אחד ויחיד** של קבוצה ב'. התאמה זו נקראת פונקציה מקבוצה א' לקבוצה ב'. קבוצה א' נקראת **התחום** של הפונקציה, וקבוצה ב' נקראת **הטווח** של הפונקציה. איבר אחד בתחום נקרא **ארגומנט** (Argument), אשר בשפות התכנות, זהו הקלט שעליו מתבצע החישוב. תוצאת החישוב של הפונקציה, הינה **ערך הפונקציה** (Function Value).

פונקציה ניתנת לתיאור באופנים שונים: באופן מילולי, על-ידי טבלה, על-ידי הצגה גראפית, על-ידי נוסחה ועוד.

הפונקציות המתמטיות מתוארות בדרך כלל על-ידי נוסחה. בחישוב הנוסחה מותאם לאיבר מסוים בתחום, לארגומנט, ערך אחד ויחיד. פונקציה יכולה גם להתאים עבור זוג ארגומנטים, או יותר, ערך אחד ויחיד. לדוגמה, פעולות החשבון הרגילות הן פונקציות שכאלה.

כאשר טווח הפונקציה מכיל איבר אחד ויחיד, ערך הפונקציה הוא קבוע, ולא תלוי בארגומנט. פונקציה כזו נקראת **פונקציה קבועה**.

לפונקציה קבועה אין צורך בקלט, כי ערכה אינו תלוי בו. מאוחר יותר נכיר פונקציה לא-קבועה, אשר אינה דורשת קלט כלשהו.

תוצאת החישוב של הפונקציה מהווה ערך, ולכן ניתן להשתמש בה גם בתוך ביטוי חשבוני. כאשר פונים לפונקציה יש לומר למחשב מה לעשות בערך המתקבל מחישוב הפונקציה. כך, הפלט של פונקציה מהווה ערך שיש לפעול עליו בעזרת הוראה כלשהי, בעוד שהפלט של הליך מודפס על המסך או במדפסת.

פונקציות מתמטיות

בסעיף זה נכיר מספר פונקציות מתמטיות, אשר אפשר להשתמש בהן בצורה פשוטה בתוך תכנית. הן מקבלות נתון אחד או יותר כקלט, ומבצעות חישוב שתוצאתו הוא **ערך אחד בלבד**.

הפניה אל הפונקציה נעשית על-ידי ציון שמה, בצירוף ערך או ערכים הנדרשים לצורך החישוב. כקלט לפונקציות מתמטיות ניתן להשתמש בכל דבר בעל ערך מספרי - קבוע, משתנה, ביטוי חשבוני, או פונקציה אחרת.

SUM a b

פונקציה זו מקבלת שני נתונים a ו-b ומחשבת את סכומם.

לדוגמה: ?PR SUM 25 70

המחשב ידפיס: 95

ניתן להפעיל את הפונקציה על מספר רב יותר של נתונים, אם נתחום אותה בין שני סוגריים עגולים -

לדוגמה: ?PR (SUM 1 2 3 4 5 6 7 8 9 10)

במקרה הזה נקבל את סכום כל הנתונים: 55

PRODUCT a b

הפונקציה PRODUCT מקבלת שני נתונים ומחשבת את מכפלתם.

לדוגמה: ?PR PRODUCT 230 11

המחשב ידפיס: 2530

גם פונקציה זו פועלת על מספר רב יותר של נתונים, באותה דרך כמו ב-SUM. כלומר, יש להציג את שם הפונקציה ואת כל הנתונים בין סוגריים עגולים.

לדוגמה: ?PR (PRODUCT 1 2 3 4 5)

נקבל: 120

DIFFERENCE a b

(לא קיימת בגירסה של APPLE LOGO)

פונקציה זו מחשבת את התוצאה של "a פחות b".

לדוגמה: ?PR DIFFERENCE 100 57

נקבל: 43

QUOTIENT a b

ב-IBM LOGO וב-APPLE LOGO II הפונקציה מבצעת את פעולת החילוק של a ב-b, ומקבלים שלם וחלק עשרוני.

לדוגמה: ?PR QUOTIENT 48 5

נקבל: 9.6

ב-APPLE LOGO הפונקציה מחשבת את חלוקת a ב-b, ומקבלים את החלק השלם בלבד של המנה.

לדוגמה: ?PR QUOTIENT 48 5

המחשב ידפיס: 9

INTQUOTIENT a b

(פונקציה זו קיימת רק ב-APPLE LOGO II)

הפונקציה מחזירה את **המנה** של חלוקת a ב-b.

לדוגמה: ?PR INTQUOTIENT 15 6

נקבל: 2

POWER a b

(פונקציה זו קיימת רק ב-IBM LOGO)

הפונקציה POWER מקבלת שני נתונים ומחשבת את התוצאה הבאה: הנתון הראשון מועלה בחזקה של הנתון השני.

לדוגמה: ?PR POWER 7 3

נקבל: 343

INT n

הפונקציה INT מקבלת ערך אחד של מספר עשרוני, ומחשבת את חלקו השלם. לפניך מספר דוגמאות:

```
?PR INT 6.99    --->   6
?PR INT 6.49    --->   6
?PR INT -6.49   --->  -6
?PR INT -6.99   --->  -6
```

n יכול להיות גם כל ביטוי חשבוני שהוא.

לדוגמה: ?PR INT (76 / 8)

נקבל: 9

REMAINDER a b

הפונקציה REMAINDER מקבלת שני נתונים a ו-b, ומחשבת את השארית של החלוקה של a ב-b.

לדוגמה: ?PR REMAINDER 16 3

הפלט יהיה: 1

מכיון שהחלוקה של 16 ב-3 שווה 5 נקבל כתוצאה את השארית 1.

נוכל להשתמש בפונקציה REMAINDER כדי לקבל את סיפרת העשרות של מספר בעל שלוש ספרות. ניקח למשל את המספר 372.
תחילה נשתמש בפונקציה INT כדי לקבל את הערך השלם של חלוקת 372 ב-10 ונקבל 37. אחר כך נשתמש בפונקציה REMAINDER כדי לקבל את השארית של חלוקת 37 ב-10:

נכתוב: ?PR REMAINDER (INT 372 / 10) 10

ונקבל: 7

ב-APPLE LOGO נוכל לרשום פקודה משולבת:

```
?PR REMAINDER (QUOTIENT 372 10) 10
```

כפי שניתן לראות,
ב- QUOTIENT: a= 372 ו-b= 10,
וב-REMAINDER: a= (QUOTIENT 372 10) ו-b= 10.

בשתי פונקציות אלו, b **אינו** יכול להיות **אפס**, כי החלוקה באפס אינה מוגדרת.

SQRT n

הפונקציה SQRT מקבלת נתון בעל ערך חיובי, ומחשבת את השורש הריבועי שלו.

לדוגמה: ?PR SQRT 64

יודפס: 8.

נתון הקלט יכול להיות גם ביטוי חשבוני, או פונקציה.

ב-APPLE LOGO הערך המחושב על-ידי SQRT הינו תמיד מספר עשרוני. לכן, תופיע בו הנקודה העשרונית, גם אם חלק השבר העשרוני שווה לאפס.

נחשב לדוגמה את היתר של משולש ישר זווית, כאשר נתונים הניצבים: a=3 ו-b=4. נוכל לכתוב לפי משפט פיתגורס את ההוראה הבאה:

```
?PR SQRT ( ( PRODUCT 3 3 ) + ( PRODUCT 4 4 ) )
```

ונקבל: 5.

שים לב, כי SQRT פועלת על ערך אחד בלבד, שהוא תוצאה של הביטוי המורכב הכלול בתוך הסוגריים החיצוניים.

ROUND n

פונקציה זו מקבלת ערך אחד ומעגלת אותו למספר השלם הקרוב אליו.

עבור n>0: העיגול נעשה כלפי מעלה, אם החלק העשרוני גדול או שווה ל-0.5. אם החלק העשרוני קטן מ-0.5, אזי העיגול נעשה כלפי מטה.

עבור n<0: העיגול נעשה כלפי מטה, אם החלק העשרוני גדול או שווה ל-0.5. כאשר החלק העשרוני קטן מ-0.5, העיגול נעשה כלפי מעלה.

להלן כמה דוגמאות:

```
?PR ROUND 12.499      --->   12
?PR ROUND 12.5        --->   13
?PR ROUND 0.7         --->    1
?PR ROUND -8.32       --->   -8
?PR ROUND -8.5        --->   -9
```

בשל הסימטריה הקיימת על ציר המספרים משני צדי האפס נמצא כי:

ROUND -:X שווה ל- -ROUND :X

כעת נכתוב הוראה, אשר מעגלת מספר עשרוני עד שתי ספרות אחרי הנקודה.

למשל: ?PR (ROUND 34.5469 * 100) / 100

נקבל: 34.55

בנוסף לפונקציות המובאות מעלה נמצא בלוגו גם פונקציות לחישובים טריגונומטריים: **SIN n**, **COS n** ו-**ARCTAN n**, אשר לא נדון בהן כאן.

* * * * *

תרגילים

1. כתוב הליך המקבל מספר כלשהו, בודק האם הוא זוגי או אי-זוגי, ומדפיס אותו עם הודעה מתאימה.
   רמז: מספר זוגי מתחלק ב-2 ללא שארית.

2. כתוב הליך המקבל מספר כלשהו למשתנה :X, בודק האם הוא מתחלק בנתון :Y ללא שארית, ומדפיס את המספר והודעה "כן/לא" בהתאם לתוצאה.

3. כתוב תכנית המקבלת למשתנה NUM: מספר כלשהו בעל שלוש ספרות, ומדפיסה בזה אחר זה:
- את סיפרת המאות,
- את סיפרת העשרות,
- את סיפרת היחידות.

4. כתוב מחדש את תרגיל מס' 3 על מנת לקבל את המספר בחלוקה של **מאות** (סיפרת המאות עם שני אפסים), **עשרות** (סיפרת העשרות עם אפס אחד), ויחידות.

5. יש להגדיר את ההליך שבשאלה 4, כך שידפיס את הפלט לפי הדוגמה הבאה: NUM: = יחידות + עשרות + מאות
לדוגמה: 500 + 30 + 9 = 539

6. נתונה מכונה למכירת ממתקים, המקבלת כתשלום מטבע של שקל אחד, ומחזירה עודף במטבעות של 1/2 שקל, 10, 5 ו-1 אג'.

כתוב תכנית המקבלת כנתון את מחיר הממתק, מחשבת ומדפיסה את העודף במספר מינימלי של מטבעות. אם מחיר הממתק הוא 27 אג' ושולם עבורו 1 ש"ח, הפלט יהיה, למשל:

```
1 COIN(S) OF 1/2 SHEKEL.
2 COIN(S) OF 10 AGUROT.
0 COIN(S) OF 5 AGUROT.
3 COIN(S) OF 1 AGURA.
```

7. כתוב תכנית המקבלת כקלט למשתנה TIME: משך זמן בשניות, ומדפיסה זמן זה בשעות, דקות ושניות.

לדוגמה - אם הקלט הוא 16370, אז הפלט יהיה ---> 4:32:50.
המספר 4 מציין את השעות, 32 מציין את הדקות ו-50 - את השניות.

8. הגדר תכנית המורכבת משני הליכים, האחד - YAMIM והשני - EIZEYOM. התכנית מחשבת איזה יום בשבוע חל NUM: ימים אחרי יום DAY:, ומדפיסה את שם היום באנגלית.

הדרכה: הגדר הליך YAMIM, שבו המשתנה DAY: מקבל ערך מספרי המציין את היום לפי מספרו הסידורי בשבוע:

1 = יום ראשון

2 = יום שני

...

7 = יום שבת

המשתנה NUM: מקבל ערך המצביע על תקופה מסוימת המבוטאת במספר ימים.

ההליך YAMIM פונה להליך EIZEYOM המוגדר עם המשתנה YOM: ומעביר ל-YOM: ערך של ביטוי חשבוני המחשב את היום החל NUM: ימים אחרי יום DAY:. ערך זה יכול להיות 0, 1, 2,...,6.

ההליך EIZEYOM מדפיס את שם היום באנגלית בהתאם לערך של YOM:

- אם YOM: שווה 0 יודפס SATURDAY
- אם YOM: שווה 1 יודפס SUNDAY

וכן הלאה.

כאשר נרשום לדוגמה ---> ?YAMIM 2 23

נקבל: THE DAY IS WENDESDAY

הנתונים הם:

- DAY: מקבל את הערך 2,
- NUM: מקבל את הערך 23.

התוצאה, היום 'WENDESDAY' מתקבלת כדלהלן: שארית של החלוקה של הסכום 'NUW: + DAY:' במספר 7 (מספר הימים בשבוע).

9. הגדר הליך לפתרון המשוואה הריבועית: $ax^2+bx+c=0$ עבור $a<>0$ ו-$b^2-4ac>=0$. בכל מקרה אחר יש להדפיס הודעת שגיאה. השתמש בנוסחה:

$$X_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

* * * * *

נשתמש בכמה מהפונקציות שלמדנו כדי לבנות הליכים למטרות שונות.

חישוב מספרים ראשוניים

נגדיר הליך המקבל מספר נתון N:, בוחן אם הוא מספר ראשוני או לא, ומדפיס הודעה מתאימה.

זכור, כי מספר ראשוני הוא מספר המתחלק רק ב-1 ובעצמו ללא שארית. כלומר, אם קיים מספר הנמצא בין 1 ל-N:, ו-N: מתחלק בו ללא שארית, אזי N: הוא מספר **לא** ראשוני.

הגדרת PRIME

```
TO PRIME :N :X
IF :N = :X [ (PR :N [IS RISHONI ]) STOP ]
IF REMAINDER :N :X = 0 [ (PR :N [ IS NOT RISHONI ]) STOP ]
PRIME :N :X + 1
END
```

למשל, לבדיקה אם המספר 127 הוא ראשוני נרשום: ?PRIME 127 2

דרך החישוב:

- יש לערוך את הבדיקה עבור המספרים המחלקים, שבין 2 ל-N: (לא כולל N:). אם המספר לא מתחלק באף לא אחד מן המספרים האלה,

וערכו של המחלק הפך להיות שווה ל-N:, המשמעות היא, כי N: הוא ראשוני, יש להדפיס הודעה מתאימה ולסיים.

```
IF :N = :X [(PR :N [IS RISHONI] ]) STOP]
```

- אם השארית של חלוקת N: ב-X: היא אפס, המשמעות היא ש-N: לא ראשוני. הודעה על כך תודפס וההליך ייעצר:

```
IF REMAINDER :N :X = 0 [(PR :N [IS NOT RISHONI]) STOP]
```

- אם השארית איננה אפס כלומר, התנאי לא מתקיים, הביצוע מתקדם להוראה הבאה הקוראת למחזור הבא. במחזור זה X: גדל ב-1, על-מנת להמשיך את הבדיקה לשאר הערכים. הביצוע מסתיים כאשר מתקיים התנאי X:=N:. יש להדפיס הודעת הסבר.

הליך זה נכון, אך מבצע בדיקות מיותרות. ניתן להקטין לכדי מחצית את מספר הבדיקות, אם תנאי הסיום עבור המספר הראשוני יהיה:

IF :X = : N   במקום   IF :X > (:N / 2)

אפשר לעשות זאת, מכיון שהמחלק הקטן ביותר הוא 2, והמספר הנכפל בו על מנת לקבל את N: הוא הערך השווה ל-N/2:.

האם לא ניתן לצמצם את מספר הבדיקות לפחות מאשר חצי? מתברר שכן.

הבה נבחר מספר N כלשהו ונמצא כי אם $X>\sqrt{N}$ וגם $Y>\sqrt{N}$, אזי $N<X*Y$.
אם N איננו ראשוני יש לו שני מחלקים, שאחד מהם **חייב** להיות קטן מהשורש של N. לכן, יש להגיע למספר הגדול מן השורש של N על מנת להסיק שאין לו מחלקים, ולכן ה-N הוא כן מספר ראשוני.
נוכל לרשום זאת כך:

```
IF SQRT :N < :X [(PR :N [IS RISHONI ])]
```

פעולת הכפל מהירה בדרך כלל מפעולת השורש במחשבים. ננצל מידע זה ונרשום:

```
IF :N < :X * :X [(PR :N [IS RISHONI ])]
```

בהתאם לכך נגדיר את ההליך PRIME1, בצורה היעילה ביותר שלמדנו עד כה.

הגדרת PRIME1

```
TO PRIME1 :N :X
IF :N < :X * :X [(PR :N [IS RISHONI] ) STOP ]
IF REMAINDER :N :X = 0 [(PR :N [IS NOT RISHONI]) STOP ]
PRIME1 :N :X+1
END
```

עוד ניתן לשפר ולהפחית לחצי ממספר הבדיקות שהגענו אליהן עד כה. ידוע, שכל מספר המתחלק במספר זוגי מתחלק גם ב-2. על-כן, אם הוא לא מתחלק ב-2, ברור שהוא לא יתחלק במספר זוגי אחר כלשהו, ולכן נוכל לדלג בבדיקה על כל המספרים הזוגיים.

לצורך זה נגדיר הליך PRIME2 הבודק תחילה אם ערכו של N: הוא 2. אם כן, תודפס ההודעה ש-N: הוא מספר ראשוני והתכנית תעצור. אחרת, יבדוק ההליך אם N: הוא מספר זוגי.

- אם כן, תודפס הודעה ש-N: הוא לא ראשוני והביצוע יסתיים.
- אם לא, יפנה PRIME2 לתת-הליך PRIME3, אשר מבצע בדיקה של מספרים אי-זוגיים בלבד. הוא יתחיל במחלק הראשון שהוא 3.

הגדרת PRIME2

```
TO PRIME2 :NUM
IF :NUM = 2 [PR :NUM [IS RISHONI] STOP]
IF REMAINDER :NUM 2 = 0 [(PR :NUM [IS NOT RISHONI]) STOP ]
PRIME3 :NUM 3
END
```

הגדרת PRIME3

```
TO PRIME3 :N :X
IF :N < :X * :X [(PR :N [IS RISHONI] ) STOP ]
IF REMAINDER :N :X = 0 [(PR :N [IS NOT RISHONI ]) STOP ]
PRIME3 :N :X+2
END
```

כשנרצה לבדוק אם המספר 127 הוא ראשוני, נרשום:

```
?PRIME2 127
```

מכיון שהמספר הנתון איננו מספר זוגי, פונה הליך זה ל-PRIME3 ומעביר ל-N: את ערכו של NUM:, ולמחלק X: הוא נותן את הערך ההתחלתי 3. בכל מחזור גדל הערך של X: ב-2.

מספר מושלם

מספר מושלם הוא מספר, אשר סכום כל המחלקים שלו ועוד 1 שווה למספר עצמו. לדוגמה, המספר 6 הוא מספר מושלם, כי סכום המחלקים שלו 2 ו-3 ועוד 1 שווה ל-6 (6=1+3+2). גם המספר 28 הוא מספר מושלם (28=1+14+7+4+2).

נכתוב עתה תכנית TO PERFECT, אשר בודקת אם מספר נתון NUM: הוא מספר מושלם, ומדפיסה הודעה בהתאם.

<u>שלבי הפתרון</u>:

א. יש לבדוק אם ערך המשתנה DVS: הוא המחלק של NUM: ללא שארית.

```
IF REMAINDER :NUM :DVS = 0
```

- אם כן, בקריאה למחזור הבא יש להוסיף למשתנה צובר SUM: את המחלק DVS: וגם את המחלק השני שערכו שווה ל-NUM/:DVS:.

```
PERFECT :NUM (:DVS + 1) (:SUM + :DVS + :NUM/:DVS)
```

- אחרת, יש להתקדם למחזור הבא, ללא שינוי ערכו של SUM:,

```
PERFECT :NUM (:DVS + 1) :SUM
```

בהליך זה ערך הקלט 'SUM:' של המחזור הבא תלוי בקיום תנאי כלשהו. לכן יש לכתוב את הקריאה לרקורסיה **לכל אחת** מהאפשרויות, ובכל אפשרות לחשב את ערך הקלט בהתאם לדרישות.

בשני המקרים הערך של DVS: גדל ב-1, בעת הקריאה למחזור הבא. בקריאה להליך במחזור הראשון, המשתנה SUM: מקבל 1 כערך התחלתי, והמשתנה DVS: מקבל 2.

ב. בתום התהליך, כאשר DVS: מקבל ערך הגדול מהשורש של NUM:,

```
IF :NUM < :DVS * :DVS
```

יש לבדוק אם SUM: שווה ל-NUM:, ולהדפיס הודעה בהתאם:

```
IF :NUM = :SUM [(PR :NUM "IS "PERFECT)]              !
               [(PR :NUM [IS NOT])]
```

אחרי ההדפסה יש לעצור.
שים לב לכך, שאפשר לכתוב את ההוראה להדפסה בשני אופנים שונים.

משפט זה מורכב מהוראת תנאי מקוננת, ההופכת אותו למשפט מורכב וארוך. רצוי אם כן, להגדיר תת-הליך BDIKA, שאליו נפנה לפני העצירה, ואשר בו תתבצע בדיקת ההשוואה בין SUM: לבין NUM:. כך נוכל לרשום:

```
IF :NUM < :DVS * :DVS [BDIKA STOP]
```

הגדרת BDIKA

```
TO BDIKA
TEST :NUM = :SUM
IFT [(PR :NUM "IS "PERFECT)]
IFF [(PR :NUM [IS NOT PERFECT]]
END
```

שים לב: BDIKA הוא תת-הליך ברמה נמוכה מזה הפונה אליו, לכן ניתן להתייחס אל SUM: ו-NUM: ללא הצורך בהעברת פרמטרים.

* * * * *

תרגילים (המשך)

10. כתוב תכנית TO PRIMES להדפסת כל המספרים הראשוניים מ-2 עד 100.

11. הגדר הליך הפונה ל-PERFECT שבתרגיל 11, לבדיקה של המספרים מ-450 עד 500, כדי למצוא מספר מושלם. יש להדפיס הודעה מתאימה לכל אחד מהמספרים אשר נבדקים.
הערה: הביצוע דורש זמן מחשב רב, סבלנות!

12. כתוב תכנית שתדפיס את כל המספרים בני שלוש ספרות, אשר סכום החזקות השלישיות של כל אחת מהספרות שלהם הוא המספר עצמו.
למשל, המספר 153 ממלא דרישה זו:

$$1^3 + 5^3 + 3^3 = 153$$

13. כתוב תכנית שתדפיס את כל השלשות של מספרים שלמים, אשר יכולים ליצור משולש ישר-זווית. חשב את השלשות, אשר בהם אורך היתר הוא בתחום 2 עד 50.
לדוגמה, בשלושת המספרים 3, 4, 5 אורך היתר הוא 5.

14. כתוב הליך המקבל מספר שלם X: הגדול מ-1 ומדפיס אותו. אחר כך הוא מדפיס את המספרים המתקבלים כתוצאה מן הפעולות הנעשות על X:. תיאור ההליך:
א. מדפיס את X:.
ב. אם X: שווה ל-1, מפסיק את הביצוע.
ג. אם X: הוא מספר אי-זוגי, מתקדם למחזור הבא כאשר X: מקבל ערך השווה ל-X*3+1:. מדפיס את הערך החדש של X:.

ד. אם X: הוא זוגי, מתקדם למחזור הבא כאשר X: מחולק ב-X/2(2:). מדפיס את הערך החדש של X:.

לדוגמה, אם הקלט הוא 17 - נקבל את הפלט הבא:

17 52 26 13 40 20 10 5 16 8 4 2 1

בדוק אם תמיד מתקבלת סידרה סופית (המגיעה עד למספר 1), ואם יש מקרים שבהם התכנית איננה מגיעה לידי עצירה.

* * * * *

חישובים אקראיים

לפעמים נרצה שהמחשב יבחר מספר אקראי, שאין אפשרות לחזותו מראש. באמצעות פעולה זו ניתן לכתוב תכנית שתבצע הדמייה של הטלת מטבע למשל, או הטלת קוביה, או שתחקה קליעה במספר כלשהו בגלגל ההימורים (רולטה).

בלוגו קיימת הוראה לבחירת מספר שלם באופן אקראי מתוך תחום מספרים מוגדר.

הפונקציה RANDOM

הפונקציה ה RANDOM פולטת ערך מספרי שלם גדול או שווה לאפס, אך קטן מ-ה, המוגדר כערך גבולי. הפונקציה דורשת קלט ה בעל ערך מספרי שתפעל עליו. ערך זה יכול להיות מבוטא על-ידי קבוע מספרי, משתנה מספרי, או ביטוי חשבוני.

כאשר נרשום ---> ?PR RANDOM 10

נקבל על המסך מספר X כלשהו מתחום המספרים שלהלן: (0,1,2,3,4,5,6,7,8,9), כי צריך שיהיה 0≤X<10. שים לב שהאפס כלול, אך ה-10 לא.

אם נרשום ---> `?PR RANDOM (75 / 6)`

המספר שיבחר יהיה בתחום המספרים (11, ... ,0,1,2), כי הערך של ה הוא התוצאה של (75/6) ושווה 12.5. **הערך הגבולי** השלם בחישוב זה הוא 12, ולכן 12 איננו כלול בתחום.

אם נרצה לבחור את אחד הערכים מתחום המספרים של גלגל המזלות שהוא (0,1,2,...,40), נרשום ערך גבולי 41, אשר לא נכלל בתחום:

נרשום ---> `?PR RANDOM 41`

בהטלת קוביה, הערכים האפשריים הם (1,2,3,4,5,6), ולכן נשתמש במספר 7 כגבול עליון:

נכתוב ---> `?PR RANDOM 7`

כאן יש שגיאה, כי תחום המספרים בביטוי זה כולל את האפס '0', אשר איננו בתחום המספרים **של הקוביה**. נוכל לפתור בעיה זו אם **נוסיף 1** לתוצאת החישוב שתתקבל, כלומר

אם התוצאה היא 0 נקבל 1,
אם התוצאה היא 1 נקבל 2,
אם התוצאה היא 2 נקבל 3,
אם התוצאה היא 3 נקבל 4,
אם התוצאה היא 4 נקבל 5,
אם התוצאה היא 5 נקבל 6,
אם התוצאה היא 6 נקבל... לא !!
אסור ש-6 יהיה כלול בתחום, אחרת נקבל 7.

אם כן, נרשום ---> `?PR 1 + RANDOM 6`

הערך 1 מתוסף לתוצאה המתקבלת מפעולת RANDOM על הנתון 6.
הערך הגבוה ביותר יכול להיות 5, ובתוספת 1, נקבל 6.

אפשר להציג זאת גם כך --->

```
?PR (RANDOM 6) + 1
```

הסוגריים מיועדים לקבוע עדיפות וכך, 1 מתוסף לתוצאה אשר חושבה תחילה על-ידי 'RANDOM 6'. בהעדר סוגריים, הביטוי 1 + 6 שהוא 7 היה מתקבל כקלט לפקודה RANDOM.

לעתים מתעורר הצורך לבחור מספרים מתחום.כלשהו, נאמר מ-12 ועד 15 (כולל), כך ש-12 הוא הערך המינימלי שבתחום - 'MIN', ו-15 הוא הערך המקסימלי שבו - 'MAX'. המספר הקטן ביותר מפעולת RANDOM הוא אפס, אם נוסיף לו 12, נשנה את הערך המינימלי ל-12. ואז, אם התוצאה היא 1 נקבל 13, אם היא 2 נקבל 14 ואם היא 3 נקבל 15, אך עד כאן. כלומר, על הפעולה ליצור מספרים מהתחום הבא (0,1,2,3). הערך הגבולי יהיה במקרה זה - 4.

אם כן, נרשום --->

```
?PR 12 + RANDOM 4
```

ערך גבולי בבעיות כאלו הוא **מספר** הערכים הנמצאים בתחום המוגדר. מקבלים מספר זה אם מפחיתים מערך ה-MAX את ערך ה-MIN ומוסיפים 1.

לפי הדוגמה לעיל: 15-12+1=4

נגדיר הליך RNDMIS לבחירת מספר אקראי מכל תחום שהוא, כאשר משתנה MAX: מייצג את הערך המקסימלי והמשתנה MIN: מייצג את הערך המינימלי.

הגדרת RNDMIS

```
TO RNDMIS :MAX :MIN
PR :MIN + RANDOM (:MAX - :MIN + 1)
END
```

RANDOM היא פונקציה, דהיינו הוראה הפולטת ערך. לכן, יש להורות למחשב מה לעשות בערך המופק על-ידי הפעלתה. ערך זה יכול לשמש כחלק מביטוי חשבוני, או להיות קלט למשתנה של הליך מוגדר בעת

הפניה אליו. פונים להליך בפקודה ישירה מתוך המצב הרגיל של לוגו, מתוך הליך אחר כחלק מההוראות שלו, וגם מתוך ההליך עצמו באופן רקורסיבי.

אחד ההליכים שהגדרנו בתחילת הספר הוא MALBEN1, המחשב את ההיקף והשטח של מלבן כלשהו. בעת הקריאה להליך הוא קיבל כקלט שני ערכים למשתנים ROCHAV: ו-ORECH:. באמצעות השימוש ב-RANDOM נוכל ליצור ערכים אקראיים בתחום המספרים (25..50) למשל, ולהשתמש בהם לציון צלעות המלבן.

נרשום ---> ?MALBEN1 (25 + RANDOM 26) (25 + RANDOM 26)

כתוצאה מפקודה זו הפונקציה RANDOM תפעל פעמיים: המספר האקראי שייבחר מחישוב הביטוי הראשון יהיה קלט למשתנה הראשון ROCHAV:, ומהביטוי השני נקבל מספר אקראי אחר למשתנה השני ORECH:.

להלן דוגמה להליך היוצר N: מספרים אקראיים בתחום (20..30) ומדפיס את סכומם.

הגדרת SUMRND

```
TO SUMRND :N :SUM
IF :N < 1 [PR :SUM STOP]
SUMRND :N-1 (:SUM + 20 + RANDOM 11)
END
```

בעת הקריאה להליך נקבע ל-SUM:, המשמש כמשתנה צובר, את הערך אפס. בכל מחזור מתוסף לו מספר אקראי חדש:

```
:SUM + 20 + RANDOM 11
```

התוצאה מהווה קלט למחזור הבא.

משתנה N: המשמש משתנה מונה מציין את מספר הערכים הדרוש. בכל מחזור מופחת ממנו 1, בהתאם לשיטת הספירה לאחור. תנאי הסיום הוא N=0:, כאשר הסתיימה בחירת כל N: הערכים האקראיים וצבירתם במשתנה SUM:. בתום הביצוע מודפס ערך הסכום (PR :SUM), וההליך יסתיים בהוראת העצירה STOP.

נציג תכנית היוצרת NUM: מספרים אקראיים בתחום (0..20) ומדפיסה רק את אלה הגדולים מ-10. בביצוע זה נדרשות שתי פעולות ביחס למספר הנבחר: בדיקה אם הוא גדול מ-10 והדפסה שלו.

אנו למדים שיש צורך בקישור הערך האקראי עם משתנה כלשהו, אשר עליו נוכל לבצע יותר מחישוב אחד. הערך של המשתנה הוא קבוע, ונשמר כל עוד לא ניתנה הוראה לשינוי. ערך זה אובד עם סיום ביצוע ההליך.

פתרון זה דורש שני הליכים:

- הליך TO TEN מוגדר עם משתנה, אשר מבצע את הבדיקה. אם התנאי מתקיים הוא מדפיס את התוצאה.
- הליך TO TENRND פונה NUM: פעמים ל-TEN. בכל פניה המשתנה המוגדר ב-TEN מקבל מספר אקראי אחר. מספר זה יש לבדוק ולהדפיס לפי תנאי הבעיה.

הגדרת TENRND

```
TO TENRND :NUM
REPEAT :NUM [ TEN (RANDOM 21) ]
END
```

הגדרת TEN

```
TO TEN :MIS
IF :MIS > 10 [PR :MIS]
END
```

כאשר רושמים ---> ?TENRND 15

TENRND פונה 15 פעמים ל-TEN. בכל פניה המשתנה MIS: מקבל מספר אקראי חדש, שהוא הערך המתקבל מביצוע 'RANDOM 21' בכל קריאה.
TEN בודק אם הערך של MIS: גדול מעשר:

```
IF :MIS > 10
```

אם כן - הוא מדפיס אותו. אחרת, הוא לא מבצע דבר.

להלן דוגמה לפתרון בעיה לחישוב ממוצע של NUM: מספרים אקראיים בתחום (0..10). נחזור על החישוב לכל ערך של NUM: מ-1 ועד 30. תרגיל זה הינו ניסוי של חישוב ממוצע סטטיסטי, שבו כל התוצאות שיתקבלו יהיו בערך 5.

הגדרת MEMUZA

```
TO MEMUZA :NRND :SCHUM :NUM
IF :NUM > 30 [STOP]
PR (:NRND + :SCHUM) / :NUM
MEMUZA (RANDOM 11) (:NRND + :SCHUM) (:NUM + 1)
END
```

בקריאה הראשונה ל-MEMUZA אנו קושרים למשתנה NRND: ערך של מספר, אשר נבחר באופן אקראי באמצעות הביטוי 'RANDOM 11'. כתוצאה מפעולה זו NUM: המספרים האקראיים שנבחרו עד כה הוא '1'. מכיוון שטרם צברנו מספרים אלה במשתנה צובר SCHUM:, ערכו הוא עדיין '0'.

לכן נרשום ---> `?MEMUZA (RANDOM 11) 0 1`

בכל מחזור מודפס ממוצע המספרים שנבחרו עד כה:

`PR (:NRND + :SCHUM) / :NUM`

בכל קריאה חדשה נבחר מספר אקראי אחר,
שיהיה ערך ל-NRND: של המחזור הבא: `(RANDOM 11)`

לערך של SCHUM: מתוסף הערך של NRND:
שקיבלנו במחזור הנוכחי: `(:NRND + :SCHUM)`

ומספר הערכים שנבחרו גדל ב-1: `(:NUM + 1)`

תנאי סיום התכנית הוא כאשר `NUM > 30`

* * * * *

תרגילים (המשך)

15. כתוב הליך הבוחר ומדפיס 10 מספרים אקראיים בתחום (1..25).

16. כתוב הליך היוצר 5 מספרים אקראיים בתחום (60..100), ומדפיס את הממוצע שלהם.

17. כתוב תכנית הבוחרת N: מספרים אקראיים בתחום (0..100), בודקת ומדפיסה רק את אלה הגדולים מ-50.

18. הגדר הליך אשר ידפיס NUM: מספרים אקראיים **שליליים** בתחום (-25..-11).

    רמז: הכפל ב-(-1) את הערך המתקבל מהביטוי של הפונקציה.

19. כתוב תכנית היוצרת 2 מספרים אקראיים בתחום (27..40). בשורה הראשונה היא מדפיסה אותם לפי סדר בחירתם, ובשורה השניה - לפי סדר עולה.

20. כתוב תכנית להדמיית הטלת מטבע MIS: פעמים, כאשר '0' מציין צד אחד של המטבע, ו-'1' - את צדו השני.
    - אם '1' נבחר יותר מ-MIS/2:, יש להדפיס הודעת זכיה.
    - אם לא, יש להדפיס הודעת הפסד.

21. כתוב תכנית להדמיית MIS: זריקות של זוג קוביות. על התכנית לרשום בטור אחד את תוצאות הזריקה של הקוביה הראשונה, בטור שני את התוצאות של השניה, ובטור שלישי - את סכום שתי התוצאות.

22. כתוב תכנית ליצירת 10 תרגילי חיבור של שני מספרים אקראיים, בתחום (1..99). על המחשב להדפיס אותם ואת תוצאת החישוב שלהם.

* * * * *

הצבת הסמן במסך - SETCURSOR

נכתוב הליך המדפיס נקודות היוצרות על המסך קו אלכסון יורד משמאל לימין.

הגדרת ALEX

```
TO ALEX
CLEARTEXT
SETCURSOR [0 0] TYPE ".
SETCURSOR [1 1] TYPE ".
SETCURSOR [2 2] TYPE ".
. . . .
SETCURSOR [24 24] TYPE ".
END
```

ניתן לצמצם הליך זה, המורכב מ-25 הוראות החוזרות על עצמן, אם נהפוך אותו להליך רקורסיבי. נגדיר את ההליך עם משתנים, אשר יקבלו בכל פעם ערך חדש לציון העמודה והשורה. יש לשים לב לכך שהערכים צריכים להיות בתוך **רשימה**.

אם נרשום לדוגמה: SETCURSOR [:Y :X]

נקבל את ההודעה הבאה:

SETCURSOR DOESN'T LIKE :Y AS INPUT

אם משתנה מופיע בתוך רשימה, לוגו לא תדע מה ערכו, כי היא מקבלת אותו כנתון ולא כמשתנה בעל ערך.
זכור את ההבדל בין 'PR :X' לבין '[:X] PR'.

בלוגו קיימות פקודות לבניית רשימה, אך לפני שנלמד אותן, הבה נכיר את ההוראה המציגה דברים על המסך. זוהי הוראה הדומה ל-PRINT, אבל יש ביניהן הבדל בהצגה של רשימות.

ההוראה SHOW

הפונקציה SHOW פועלת על דבר **אחד** בלבד.

נשתמש בדוגמה: נניח כי למשתנה X: יש ערך 100.

אם נרשום --->	PR :X	המחשב ידפיס --->	100
ואם נרשום --->	SHOW :X	נקבל גם כן --->	100

במקרה זה אין הבדל וקיבלנו אותו פלט. אבל,

אם נרשום --->	PR [:X]	המחשב ידפיס --->	:X
ואם נרשום --->	SHOW [:X]	נקבל --->	[:X]

כלומר, SHOW מציגה את **הרשימה** כפי שהיא.

עוד דוגמה --->
```
?PR [I LIKE LOGO]
```
נקבל את המשפט:
```
I LIKE LOGO
```

אבל כאשר נרשום --->
```
?SHOW [I LIKE LOGO]
```
המשפט יוצג בתוך רשימה:
```
[I LIKE LOGO]
```
נלמד כעת הוראת בניה, אשר מאחדת, או מצרפת, דברים לרשימה אחת.

עריכת רשימה - הפונקציה SENTENCE (קיצור - SE)

המבנה הכללי: **SE דבר**

או: **SE דבר1 דבר2**

הפונקציה SE מקבלת דבר אחד או שני דברים כקלט, **ופולטת רשימה** המכילה דברים אלה (ב-APPLE LOGO II הפונקציה SE פועלת על 2 דברים, אך לא על דבר אחד). אם ה"דבר" הינו **מספר** או **מילה** (רצף

של תווים), ערך הפלט שלה הוא רשימה המכילה את המספר עצמו, או את המילה עצמה.

אם נרשום ---> ?SHOW SE 57

נקבל מספר זה בתוך רשימה: [57]

אך אם נרשום ---> ?SHOW SE "DONALD "DUCK

נקבל את שתי המילים ברשימה אחת: [DONALD DUCK]

אם ה"דבר" הינו **משתנה**, SE תציב את **ערכו** של המשתנה ברשימה שהיא בונה.
אם ה"דבר" הוא **רשימה**, SE תצרף את האיברים לרשימה שהיא בונה.

ניתן לאחד **מספר דברים** באמצעות ההוראה SE, אם נתחום את כל ההוראה בסוגריים **עגולים**:

```
--------------------------------
( SE 1דבר 2דבר ... דברn )
--------------------------------
```

נגדיר הליך המדגים כיצד לאחד דברים באמצעות SE, כלומר **לצרף** אותם יחד.

הגדרת MISHPAT

```
TO MISHPAT :MIS :MILA
SHOW SE :MIS :MILA
SHOW SE [ELY] [GIL AVI]
SHOW (SE :MILA [POPEY & OLIVE] "HELLO :MIS)
END
```

כאשר נרשום ---> ?MISHPAT 2001 "COCA_COLA

בשורה הראשונה יודפס: [2001 COCA_COLA]

כלומר SE איחדה את ערכו של MIS: עם ערכו של MILA: לרשימה אחת.

בשורה השניה נקבל: [ELY GIL AVI]

כאן SE איחדה את איברי הרשימה [GIL AVI] יחד עם תוכן הרשימה [ELY] ברשימה אחת.

ובשורה השלישית נקבל: [COCA_COLA POPEY & OLIVE HELLO 2001]

שים לב: מאחר שתוצאת החישוב של ההוראה SE מהווה ערך, יש להורות למחשב מה לעשות בערך זה. בדוגמה שלמעלה השתמשנו בהוראה SHOW.

כעת נשתמש בפונקציה SE, על מנת להגדיר מחדש את ההליך ALEX, כדי שיפעל עם משתנים.

ההליך ALEX1

```
TO ALEX1 :Y :X
IF :Y > 24 [STOP]
SETCURSOR SE :Y :X
TYPE  ".
ALEX1 :Y+1 :X+1
END
```

ערך התחלתי של המשתנה X:, המציין את העמודה, הוא המספר '0'. גם למשתנה Y:, המציין את השורה, נציב ערך '0'.
נרשום עתה ---> ?ALEX1 0 0

הביטוי 'SE :Y :X' מאחד את הערכים של X: ו-Y: בתוך רשימה המשמשת קלט ל-SETCURSOR, כאילו רשמנו - 'SETCURSOR [0 0]'. המחשב מדפיס במקום זה את סימן הנקודה '.' ומתקדם למחזור הבא, כאשר X: גדל ב-1 וגם Y: גדל ב-1, וכך ממחזור למחזור.
תנאי הסיום הוא כאשר המשתנה Y: מקבל ערך, אשר לא מציין כל שורה שהיא על המסך (השורה 24 היא האחרונה).

<u>הערה:</u> כאשר הפונקציה SETCURSOR מקבלת קואורדינטות של נקודה שאיננה מוגדרת על המסך, מקבלים הודעת שגיאה.

שמת לב בוודאי, שערכו של X: זהה לערכו של Y: בכל מחזור. לכן ניתן להגדיר הליך זה במשתנה אחד בלבד, שישמש גם לציון העמודה וגם לציון השורה.

בשורת ההכרזה נרשום: TO ALEX2 :X

להצבת הסמן נכתוב: SETCURSOR SE :X :X

ובקריאה לרקורסיה: ALEX2 :X+1

בתרגיל הבא נגדיר הליך IXIM, המדפיס X-ים היוצרים את צורת האות 'X' על פני המסך. למעשה, צורת 'X' בנויה משני קווים אלכסוניים מצטלבים באופן סימטרי. אחד היורד משמאל לימין, והשני - מימין לשמאל. האלכסון הראשון מתחיל בנקודה [0 0], ומסתיים בנקודה [24 24] והאלכסון השני מתחיל ב-[24 0] ומסתיים ב-[0 24].

גם בפתרון זה די לנו במשתנה אחד. לאלכסון הראשון, כמו ב-ALEX2, נרשום 'SETCURSOR SE :X :X'. באלכסון השני, הערך של השורה שווה לזה של האלכסון הראשון, אבל את הקלט של העמודה יש לחשב. על פי החישוב נמצא, שערך העמודה באלכסון השני מתקבל אם נפחית מהמספר 24 את ערך העמודה של האלכסון הראשון:

- כאשר העמודה באלכסון הראשון היא '0', באלכסון השני היא '24'. ואכן: 24-0=24.
- כאשר העמודה באלכסון הראשון היא '1', העמודה באלכסון השני היא '23' (24-1=23).
- כאשר העמודה באלכסון הראשון היא '2', העמודה באלכסון השני היא '22' (24-2=22).

  וכן הלאה...,
- כאשר העמודה האחרונה באלכסון הראשון היא '24', העמודה באלכסון השני היא '0' (24-24=0). וכאן נסיים.

לפיכך, להצבת הסמן להדפסת האלכסון היורד מימין לשמאל נרשום:

```
SETCURSOR SE :X (24 - :X)
```

הגדרת IXIM

```
TO IXIM :X
IF :X > 24 [STOP]
SETCURSOR SE :X :X
TYPE "X
SETCURSOR SE :X (24 - :X)
TYPE "X
IXIM :X + 1
END
```

* * * * *

תרגילים (המשך)

23. הגדר הליך המדפיס אותיות Z-ים, היוצרות על המסך את הצורה של האות 'Z'.
חזור על בעיה זו עבור האות A, האות H, ועבור Y, V, N, K.

24. כתוב תכנית היוצרת סימולציה של "יו-יו", המחקה תנועת כדור היורד מטה ועולה מעלה לסירוגין. הגדר לצורך הפתרון שני הליכים -אחד YOYODOWN, והשני YOYOUP. השתמש באות 'O' הדומה לכדור.

25. הגדר הליך להדפסת 500 כוכבים בנקודות הנבחרות באופן אקראי.
השתמש במשפט:

```
SETCURSOR SE (RANDOM 25) (RANDOM 39)
```

<u>הסבר</u>: הפונקציה RANDOM בביטוי הראשון בוחרת ערך המציין את העמודה, ובביטוי השני היא בוחרת ערך לשורה. SE מציבה ערכים אלה ברשימה, אשר משמשת קלט ל-SETCURSOR.

<u>זכור</u>: אם הינך משתמש בגירסה של APPLE LOGO II או של APPLE LOGO, יש להפוך את סדר הערכים של הנקודה. המשפט יירשם כך:

```
SETCURSOR SE (RANDOM 39) (RANDOM 24)
```

פרק 9

מבוא לתכנות פונקציונלי

בפרק הקודם הכרנו חלק מהפונקציות השימושיות המוגדרות כחלק בלתי נפרד משפת לוגו. אולם, מה קורה אם נרצה להשתמש בפונקציה לחישוב הערך המוחלט למשל, שאיננה מוגדרת בשפה?

הגדרת פונקציות

לוגו מאפשרת למשתמש להגדיר **פונקציות** (Functions), כשם שהיא מאפשרת להגדיר **הליכים** (Procedures).

הפקודות בהליך גורמות לשינוי על המסך (CLEARTEXT ,PR), שינוי בזיכרון (REPEAT ,TEST) וכדומה. התוצאה של ההליך הינה פלט מוצג או מודפס.

פעולות החישוב בפונקציה, לעומת זאת, יוצרות ערך. דהיינו, כאשר קוראים לפונקציה, תוצאת החישוב מהווה את **ערך הפונקציה**.

פונקציה הינה תכנית, אשר בתום החישוב שהיא עורכת, היא מחזירה **ערך אחד ויחיד** הקרוי **ערך הפונקציה** (Function Value). הפונקציה מופעלת על נתון אחד או יותר (ארגומנטים - Arguments), אשר קובעים את ערכה.
בדומה להגדרת הליך, מתחילים הגדרה של פונקציה על-ידי המילה TO, לידה שם הפונקציה ולאחריו שמות המשתנים. המשתנים הם **הפרמטרים הפורמליים**, שאליהם מועברים ערכי הארגומנטים בהתאמה. בשורות שלאחר הכותרת ניתנת סידרת ההוראות המתארת את חישוב הערך של הפונקציה ומסיימים במילה END.

כותרת פונקציה: **TO (שם פונקציה) פרמטר1 פרמטר2 ... פרמטרת**

מילת המפתח בתכנית המגדירה פונקציה היא **OUTPUT**, ובקיצור **OP**.
OP מעבירה את הערך המחושב למי שקרא לפונקציה, וזהו תנאי להגדרת התכנית כפונקציה. **עם מסירת הערך לקורא מסתיים ביצוע הפונקציה.**

בין פונקציות השפה מצאנו את פונקציית הסכום 'SUM' ופונקציית השורש הריבועי 'SQRT', אך אין פונקציה להעלאה בריבוע למשל, ואין פונקציה להעלאה בחזקה שלישית.
הבה נגדיר פונקציות אלו.

<u>הגדרת CUBE</u>

(העלאת מספר בחזקה שלישית)

```
TO CUBE :A
OP :A * :A * :A
END
```

<u>הגדרת SQR</u>

(העלאת מספר בריבוע)

```
TO SQR :X
OP :X * :X
END
```

כאשר מפיקים במחשב ערך יש להורות לו מה לעשות בערך זה. CUBE ו-SQR הן פונקציות ולכן עלינו לפנות אליהן על-ידי כתיבת השם ומתן הערכים למשתנים, וגם - לציין מה לעשות בערך המתקבל כתוצאת החישוב. אם לא נעשה כן - נקבל הודעת שגיאה.

אם נרשום ---> ?CUBE 21

נקבל הערה כגון: I DON'T KNOW WHAT TO DO WITH 9261

עתה נשתמש בהוראה האומרת למחשב מה לעשות בערך המתקבל.

נוכל לרשום למשל ---> ?PR SQR 7

ונקבל: 49

היחס לפונקציה כלשהי כמוהו כאל ערך, ועל כן היא יכולה להופיע כחלק מביטוי חישובי.

לדוגמה: PR (75 + SQR 5) / 2

או: TEST (CUBE 4) > 50

או למשל: PR SUM 10 SQR 3

בפרק על משפטי התנאי הגדרנו הליך MUHLAT2, המחשב את הערך המוחלט של נתון מספרי **ומדפיס** אותו. הפעם נכתוב את אותו תהליך חישוב כפונקציה. כלומר, כתוצאה מביצוע הפונקציה יתקבל **ערך**, בניגוד להליך, שבו המחשב פולט את התוצאה על המסך. נלמד את ההבדלים בין הליך לבין פונקציה מבחינת אופן ההגדרה ואופן הביצוע.

הגדרת ABS (פונקציה)

```
TO ABS :X
IF :X < 0 [OP -:X]
OP :X
END
```

הגדרת MUHLAT2 (הליך)

```
TO MUHLAT2 :X
IF :X < 0 [PR -:X STOP]
PR :X
END
```

בהליך MUHLAT2 משמשת ההוראה STOP לסיום ההליך באחד מסעיפי ההוראה IF. הביצוע ימשיך לאחר ההוראה END. בפונקציה ABS מקבלים ב-OP את הערך המחושב ובכך **מסתיים** ביצוע הפונקציה. אין חשש שבמקרה זה ימשיך הביצוע בפקודה העוקבת. התכנית שקראה לפונקציה תטפל בערך שהתקבל.

בפנייתנו ל-MUHLAT2 נרשום ---> ?MUHLAT2 -19

ונקבל את התוצאה על המסך: 19

לא כן בפניה לפונקציה -

כאן יש לרשום הוראת הדפסה לפני ABS ---> ?PR ABS -19

כדי שהערך המוחזר אכן יוצג על המסך: 19

גם כאשר פונקציה אחת פונה לפונקציה אחרת, ההוראה OP שבפונקציה הקוראת פועלת **על התוצאה**, כלומר על **הערך**, המתקבל מהפונקציה הנקראת.

נגדיר פונקציה DIV, המקבלת שני מספרים a ו-b, ומחזירה את ערך המנה של חלוקת a ב-b (בגירסה של APPLE LOGO קיימת הפונקציה QUOTIENT למטרה זו).

הגדרת DIV

```
TO DIV :A :B
OP INT (:A / :B)
END
```

כאשר נרשום ---> ?PR DIV 14 3

נקבל: 4

OP מקבלת מ-INT את הערך שלה, ואת הערך הזה היא מעבירה.

* * * * *

תרגילים

1. הגדר פונקציה MAX, המקבלת שני מספרים ומחזירה את הערך הגדול מבין השניים. הדפס את התוצאה. עשה זאת גם בשאר התרגילים שבהמשך.

2. הגדר פונקציה MIN, הפועלת על שני מספרים ומחזירה את הערך הנמוך מבין השניים.

3. הגדר פונקציה FRAC, המחזירה את החלק העשרוני של מספר עשרוני כלשהו. אם הנתון הוא 17.5019 למשל, הערך המוחזר יהיה 0.5019.

4. בהנחה שלא נמצאת בלוגו הפונקציה REMAINDER להצגת שארית החילוק של a ב-b, הגדר אותה בעצמך. קרא לפונקציית השארית בשם 'MOD'.
<u>רמז</u> - השתמש ב-INT.

5. הגדר מחדש את הפונקציה לעיגול מספרים, כאילו לא קיימת בשפה הפונקציה ROUND. קרא לה 'RND'.

6. הגדר פונקציה NEGARAND המקבלת שני נתונים :MAX ו-:MIN, ומחזירה ערך שהוא מספר אקראי שלילי בתחום (-:MAX...-:MIN).

7. הגדר פונקציה PAI המחזירה את ערכו של PI (=3.1415).

אם קיימת הפונקציה PI בגירסת לוגו שאתה משתמש בה, הנח שהיא איננה קיימת, והגדר אותה בעצמך.

* * * * *

פונקציה רקורסיבית

ראינו שפונקציה יוצרת ערך, אשר משמש כקלט לתכנית אחרת שקראה לה, או לפונקציה אחרת שפנתה אליה.

למעשה, פונקציה יכולה לקרוא לא רק לפונקציה אחרת, אלא היא יכולה לקרוא גם לעצמה. בפונקציה כזו מתבצע החישוב במחזורים, אשר תוצאת כל אחד מהם תלויה בתוצאה של המחזור, אשר חושב לפניו. כל מחזור פועל על תוצאה שקיבל מהמחזור שנקרא על-ידו.
בהמשך יבואו הסברים והבהרות מפורטים יותר.

פונקציה מסוג זה הינה **פונקציה רקורסיבית**.

האלגוריתם של אוקלידס

בהסתמך על **האלגוריתם של אוקלידס** (מתמטיקן יווני מן המחצית השניה למאה הרביעית לפנה"ס), נגדיר פונקציה למציאת **המ**חלק **המ**שותף **המ**קסימלי '**ממם**' של 2 מספרים. באנגלית מכנים זאת: Greatest Common Divisor - GCD.

האלגוריתם (Algorithm), שהוא תהליך חישוב, מתבסס על ההגדרה הבאה: כל מחלק משותף של שני מספרים טבעיים (לא אפס) מחלק גם את הסכום וגם את ההפרש שלהם.

תיאור האלגוריתם של אוקלידס:

א. אם a=b, אזי ה-ממם שווה לערך עצמו (ל-a או ל-b).

ב. אם X הוא ה-ממם של a ו-b, אזי X הוא ה-ממם של ערכו המוחלט של הפרשם $|a-b|$.

נתון: $mX = b$,$nX = a$

מכאן ש- $a - b = nX - mX$

נציב את X מחוץ לסוגריים ונקבל $a - b = X (n - m)$

כלומר, גם ההפרש של a ו-b הוא כפולה של X. זאת, כל עוד הפרש זה גדול מאפס, כלומר כל עוד a שונה מ-b.

אם נחסר את a מ-b, הערך המוחלט של התוצאה תמיד יהיה קטן מן הערך הגדול מבין השניים. נחזור על פעולת החיסור, כאשר אחד המשתנים הוא בעל הערך הקטן מבין a ו-b, ובמשתנה השני נציב את הערך המוחלט של ההפרש שלהם, עד שערכי שני המשתנים a ו-b יהיו שווים.

לדוגמה, נתון: a=32, b=12.

נחשב את הערך המוחלט של ההפרש: $20 = |32-12|$. נציב ערך זה במשתנה בעל הערך הגדול מבין השניים, שבמקרה זה הוא המשתנה a.
הנתונים הפעם: a=20, b=12.

נחשב שוב את הערך המוחלט של ההפרש ונקבל: 8 = |20-12|. גם הפעם a הוא המשתנה בעל הערך הגדול ובו נציב את ערך ההפרש.
עכשיו נקבל: a=8, b=12.

נחזור על פעולת החיסור ונקבל: 4 = |12-8|. b הוא עתה בעל הערך הגדול, ולכן נציב בו את תוצאת החישוב.
כעת a=8, b=4.

כתוצאה מפעולת חיסור נוספת נקבל: 4 = |8-4|. הפעם ערך זה יוצב ב-a. כתוצאה מהצבה זו נקבל כי b=4 וגם a=4, כלומר שניהם שווים. לכן ממם שווה 4, על פי משפט א' של האלגוריתם.

נכתוב את האלגוריתם בשפת לוגו ונבצע את תהליך חישוב ה-ממם באופן רקורסיבי על ערכו המוחלט של |a-b|, יחד עם הערך הנמוך מבין השניים, עד ש-'a=b'.

הגדרת MMM (ממם)

```
TO MMM :A :B
IF :A = :B [OP :A]
TEST :A > :B
IFT [OP MMM (:A - :B) :B]
IFF [OP MMM :A (:B - :A)]
END
```

עבור :A ו-:B שווים, ערך הפונקציה יהיה הערך של :A והביצוע יסתיים. אחרת, תתבצע קריאה רקורסיבית באופן מותנה:

- אם ':A > :B', אז :A של המחזור הבא יקבל את ההפרש ':A-:B' ו-:B שהוא הקטן מביניהם, יעבור כפי שהוא.
- אחרת, אם :B הוא הגדול, אז :B של המחזור הבא יקבל את ההפרש ':B-:A', ואילו :A יעבור כפי שהוא.

אלגוריתם אחר עבור *ממם*, אף הוא של אוקלידס, מובא בתיאור הבא:

א. אם ערכו של אחד משני המשתנים a ו-b הוא '0', אז ערכו של השני הוא ה-*ממם*.

ב. אחרת, יש לחזור על החישוב, כאשר ערך השארית כתוצאה מחלוקת b ב-a יהיה ערך למשתנה האחד a, ואילו הערך של a יהיה ערך למשתנה השני b.

כפי שידוע, אין החלוקה באפס מוגדרת, לכן יש לבדוק אם הערך של A: הוא '0'.

- אם כן, אז ערכו של B: הוא ה-ממם, ועם מסירת ערך זה יסתיים ביצוע הפונקציה.
- אם לא, יש לחשב את ה-ממם עבור השארית והערך של A:. יש להעביר את ערך השארית למשתנה A: בקריאה למחזור הבא, ול-B: את ערך המשתנה A: של המחזור הנוכחי.

תהליך זה מתבצע באופן רקורסיבי עד שמקבלים 'A=0:'.
אלגוריתם זה נציג בפונקציה GCD:

הגדרת GCD

```
TO GCD :A :B
IF :A = 0 [OP :B]
OP GCD (REMAINDER :B :A) :A
END
```

לחישוב GCD עבור המספרים 12 ו-18, נרשום ---> ?PR GCD 12 18
ונקבל: 6

פונקציה פונה לפונקציה אחרת באמצעות OP, כי זו דרכה לפעול על הערך המוחזר מהפונקציה הנקראת.
כך גם כאשר היא פונה לעצמה: OP GCD (REMAINDER :B :A) :A

- אם A: הוא '0', הפלט יהיה הערך של B:.
- במקרה ש-B: הוא '0', כלומר אם נרשום 'PR GCD 14 0', תפנה הפונקציה באופן רקורסיבי לעצמה פעם אחת בלבד.
ערך המשתנה A: של המחזור הבא יהיה '0', כתוצאה של הביטוי 'REMAINDER 0 14', ו-B: יקבל את הערך 14.

- כאשר התנאי 'IF :A = 0' מתקיים, הערך 14 יהיה הפלט של GCD '[OP :B]', שהוא ערך התוצאה של הפונקציה.

תכניות חישוב בשברים פשוטים

נגדיר הליך **לצמצום של שבר פשוט** ונשתמש בפונקציה ממם, כשם שמשתמשים בכל פונקציה אחרת. נעשה זאת בפתרון התרגיל הבא, שבו נתונים המונה a והמכנה b. ידוע, שכדי לצמצם את השבר צריך לחלק את המונה ואת המכנה במחלק המשותף הגדול ביותר.

הגדרת ZIMZUM

```
TO ZIMZUM :MONE :MEHANE
TYPE :MONE / (GCD :MONE :MEHANE)
TYPE "/
TYPE :MEHANE / (GCD :MONE :MEHANE)
END
```

כאשר נרשום ---> `?ZIMZUM 12 18`

נקבל את הפלט הבא: `2./3.`

תחילה חילקנו את המונה '12' ב-ממם (6)

והדפסנו את התוצאה: `TYPE :MONE / (GCD :MONE :MEHANE)`

ליד זה הדפסנו את פעולת החילוק '/': `TYPE "/`

לאחר מכן חילקנו את המכנה '18' אף הוא ב-ממם

והדפסנו את התוצאה: `TYPE :MEHANE / (GCD :MONE :MEHANE)`

הערה: אם בגירסה שהינך משתמש בה מופיעה הנקודה העשרונית, ניתן למנוע את הדפסתה באמצעות פונקציית השלם INT במשפט ההדפסה:

```
TYPE INT (:MONE / (GCD :MONE :MEHANE))
```

כן גם עבור המכנה:

```
TYPE INT (:MEHANE / (GCD :MONE :MEHANE))
```

נגדיר הליך **להסבת שבר מדומה לשבר מעורב** ונשתמש לשם כך בהליך המצמצם שבר פשוט. כזכור, בשבר מדומה המונה גדול מן המכנה, ולכן הוא מכיל גם יחידות שלימות.

הגדרת MEORAV

```
TO MEORAV :MONE :MEHANE
( TYPE :MONE "/ :MEHANE "\b "=  "\b )
TEST :MONE > :MEHANE
IFT [(TYPE INT (:MONE / :MEHANE) "\b )]
IF REMAINDER :MONE :MEHANE = 0 [STOP]
(TYPE "+ "\b)
ZIMZUM (REMAINDER :MONE :MEHANE) :MEHANE
END
```

עבור המספר המדומה '70/25'

נרשום את ההוראה ---> `?MEORAV 70 25`

ונקבל את הפלט הבא: `70/25 = 2 + 4/5`

נציג תחילה את התרגיל: `( TYPE :MONE "/ :MEHANE "\b "=  "\b )`

עתה נעבור לחישוב ההליך MEORAV הבודק תחילה, אם המספר שנמסר לו הוא שבר מדומה:

```
TEST :MONE > :MEHANE
```

אם כן, הוא מדפיס את השלמים ולידם שני רווחים:

```
IFT [(TYPE INT (:MONE / :MEHANE) "\b )]
```

ההוראה שלאחר מכן מתבצעת בכל מקרה, על מנת להבטיח סיום ההליך, גם אם לא נשאר שבר פשוט לאחר הוצאת השלמים:

```
IF REMAINDER :MONE :MEHANE = 0 [STOP]
```

לדוגמה, אם המונה הוא 42 והמכנה שווה ל-14, אז התוצאה היא מספר שלם ולא מספר מעורב, לדוגמה: '42/14 = 3'.

אך אם נשאר שבר פשוט, מודפס סימן הפלוס '+', ומבוצעת קריאה ל-ZIMZUM, כדי להדפיס את השארית בצורה המצומצמת ביותר.

```
ZIMZUM (REMAINDER :MONE :MEHANE) :MEHANE
```

נגדיר הליך **לחיבור שני מספרים של שבר פשוט.** בחיבור זה לפנינו ארבעה משתנים: MONE1: MEHANE1: MONE2: MEHANE2:. ההליך מחשב את סכומם ובאמצעות MEORAV מדפיס את התוצאה בצורה המצומצמת.

ידוע ש- a/b + c/d = (ad + bc) / bd

לפי משוואה זו נפתור את התרגיל.

הגדרת HIBUR

```
TO HIBUR :MN1 :MH1 :MN2 :MH2
MEORAV ((:MN1*:MH2) + (:MN2*:MH1)) (:MH1*:MH2)
END
```

כאשר נרשום ---> ?HIBUR 3 5 7 10

נקבל את הפלט הבא: 65/50 = 1 3/10

HIBUR פונה ל-MEORAV. הוא מעביר את ערך החישוב של 'ad + bc' למשתנה MONE: ובמקרה שלנו - את הערך 65 (= 3*10 + 5*7). למשתנה MEHANE: הוא מעביר את ערך הביטוי 'bd', שהוא 50 (= 5*10).

אם נרצה בהדפסת התרגיל מתחילתו, נוסיף ל-HIBUR הוראה שתתבצע לפני הפניה ל-MEORAV:

```
( TYPE :MN1 "/ :MH1 "\b "+ "\b :MN2 "/ :MH2 "\b "= "\b )
```

ואז הפלט הסופי יהיה: 3/5 + 7/10 = 65/50 = 1 + 3/10

למעשה, כתבנו תכנית שלימה לחיבור שני מספרים של שבר פשוט.

תכנית זו בנויה לפי שיטת התכנות המבני ומורכבת מכמה הליכים, אשר אחד קורא לחברו.

- ההליך HIBUR מבצע את החישוב **ופונה** ל-MEORAV, על מנת שזה יציג את התוצאה כמספר מעורב (אם ניתן).
- ההליך MEORAV **פונה** ל-ZIMZUM, כדי לצמצם את המספר, אם הדבר אפשרי.
- ההליך ZIMZUM **פונה** לפונקציה GCD, שבאמצעותה מתבצע חישוב הצמצום.

נוכל להציג כך את התהליך:

HIBUR ——> MEORAV ——> ZIMZUM ——> GCD

* * * * *

תרגילים (המשך)

8. הגדר הליך המקבל שטח של עיגול, ומשתמש בפונקציה PAI שהגדרת בתרגיל 7, לחישוב הרדיוס של אותו עיגול.

9. הגדר הליך HISUR הפונה להליך MEORAV, לחישוב הפרש של שני מספרים של שבר פשוט.

10. הגדר הליך KEFEL לחישוב המכפלה של שני מספרים של שבר פשוט.

11. הגדר הליך HILUK לחישוב החלוקה של המספר הראשון בשני. השתמש בהליך MEORAV.

12. הגדר הליך RNDH הבוחר 4 נתונים אקראיים בתחום (1...100), שיהוו שני מספרים של שבר פשוט. ההליך פונה להליך HIBUR לחישוב והדפסה של הסכום שלהם.

13. כתוב הוראה שתבצע את תרגיל 12 עשר פעמים. כלומר, הדפסת 10 תרגילי חיבור של שני מספרים בשבר פשוט, כאשר הנתונים נבחרים באופן אקראי על-ידי המחשב.

14. השתמש בפונקציה GCD על מנת להגדיר פונקציה לחישוב המכפלה המשותפת הקטנה ביותר של שני מספרים נתונים (המכנה המשותף הקטן ביותר).

15. בהנחה שהפונקציה SQRT אינה מוגדרת בלוגו, הגדר פונקציה SQUAROOT למציאת קירוב של השורש הריבועי של מספר נתון, עד לדרגת דיוק של 0.0001.

סיוע לפתרון:

אם NUM: מהווה שטח של מלבן כלשהו, הרי ככל שההפרש בין שתי הצלעות קטן יותר, כך מתקרב המלבן לצורה של ריבוע. ידוע שצלע של ריבוע היא השורש הריבועי של שטחו.

אם נתונה צלע אחת A: של שטח NUM: כלשהו - הצלע השניה B: תהיה שווה ל-NUM/:A:.

הדרכה בהרכבת התכנית:

לפתרון התרגיל נקבע ערך התחלתי '1' עבור המשתנה A:, ואילו ל-B: ניתן את הערך NUM/:A, אשר שווה ל-NUM: עצמו.

- אם ההפרש המוחלט של A: ו-B: קטן מ-0.0001, על הפונקציה לפלוט את הערך של A:. תוכל להשתמש בפונקציה ABS לאחר שתגדיר אותה במחשב:

```
IF ABS (:A - :B) < 0.0001 [OP :A]
```

- אחרת, יש לפנות לפונקציה באופן רקורסיבי כאשר A: יקבל ערך שהוא הממוצע של שתי הצלעות. B: יקבל כאמור את הערך ':NUM/:A'.

```
OP SQUAROOT :NUM (:A+:B)/2 (:NUM/:A)
```

בתרגילים כאלה, כאשר יש להציב **ערך התחלתי** שנקבע או מחושב על-ידי האלגוריתם, נוכל להגדיר הליך ראשי אשר יפנה לפונקציית החישוב עם מתן הערכים המתאימים.

לדוגמה --->

```
TO APRXSQRT :MIS
PR SQUAROOT :MIS 1 :MIS
END
```

* * * * *

מהלך הביצוע של הליך רקורסיבי

הבה נעקוב אחר הביצוע של **תהליך חישוב** (Algorithm) בעל **מבנה רקורסיבי**, צעד אחר צעד, ותוך כדי כך נלמד על דרך פעולתו של הליך רקורסיבי.

בגמר ביצוע הוראה בתכנית כלשהי עובר המחשב להוראה העוקבת. אם הוראה **n** שבהליך AAA היא **קריאה** להליך BBB, אזי בתום ביצוע **כל ההוראות** המופיעות ב-BBB, עובר הביצוע להוראה **n+1** שבהליך AAA להמשך התהליך המוצג בתכנית.

כיצד מיושם הדבר בהליך רקורסיבי, כאשר הפניה היא אל ההליך עצמו? לאן עובר הביצוע כאשר הוא מסיים מחזור ד' למשל, שנקרא על-ידי מחזור ג' שלפניו? היכן ומתי מסתיים כל מחזור, האם בהוראה STOP, או בהוראה END?

על מנת להסביר זאת, נעזר בהליך FORWD המדפיס את המספרים מ-1 עד 3. הקלט הוא '1'.

```
TO FORWD :NUM
IF :NUM > 3 [STOP]
PR :NUM
FORWD :NUM + 1
END
```

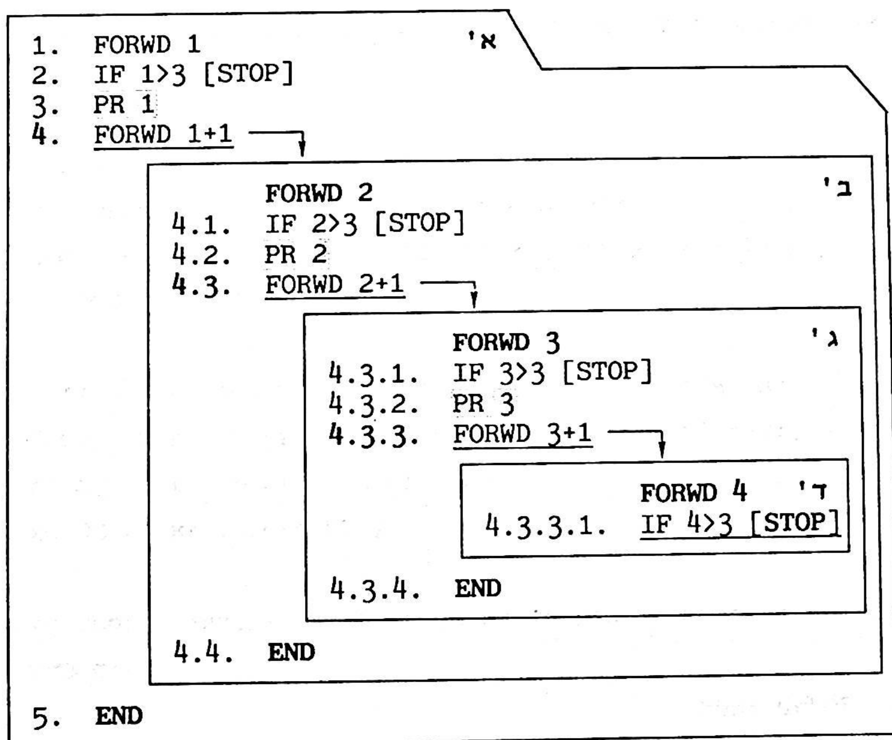
כאשר נרשום --->
נקבל את הפלט הבא:

```
?FORWD 1
1
2
3
```

תחילה הודפס המספר '1', לאחריו המספר '2' ולבסוף - '3'.

נרשום להלן את כל ההוראות המתבצעות במשך הרצת התכנית, על מנת לבדוק כל הוראה והוראה - מתי היא מתבצעת? איזו באה אחריה? מתי מסתיים כל אחד מן המחזורים, ואיך?
במקום המשתנה :NUM נכתוב את הערך עצמו בכל מחזור ומחזור, כפי שמתייחס אליו המחשב.

כאשר רושמים 'FORWD 1' ומקישים <ret> מתחיל המחשב בביצוע:



תחילה מבצע המחשב את ההוראה 1 ולאחריה את ההוראה 2. כאן אין התנאי מתקיים, לכן המחשב עובר להוראה 3, אשר מדפיסה את **המספר** 1. בהוראה 4 מתבצעת קריאה רקורסיבית להוראה 'FORWD 1+1' והמחשב עובר למחזור אחר (מחזור ב'). בתום הביצוע של **כל** ההוראות הכלולות במחזור ב' נחזור למחזור א', למילת הסיום END. במחזור ב' תופעלנה כל ההוראות המסומנות במסגרת ב' ובמסגרות הפנימיות הכלולות בו.

מחזור ב' מתחיל להתבצע כאשר הקלט שלו הוא '2'. בהוראה 4.1 התנאי לעצירה לא מתקיים, לכן ממשיך המחשב להוראה 4.2, שמדפיסה את **המספר** 2. בהוראה 4.3 נעשית פניה למחזור ג', והפעם ערך הקלט הוא '3', כי 'FORWD 2+1' (שים לב - עדיין לא הגיע ההליך למילה END שב-4.4).

בהוראה 4.3.1 שבמחזור ג' לא מתקיים התנאי, ולכן הוראה 4.3.2 מדפיסה את **המספר** 3. הוראה 4.3.3 קוראת למחזור ד' 'FORWD 3+1'.

במחזור ד', התנאי 'IF 4>3' שבהוראה 4.3.3.1 אכן מתקיים, וכתוצאה - נעצר ביצוע מחזור זה על-ידי ההוראה **STOP**.

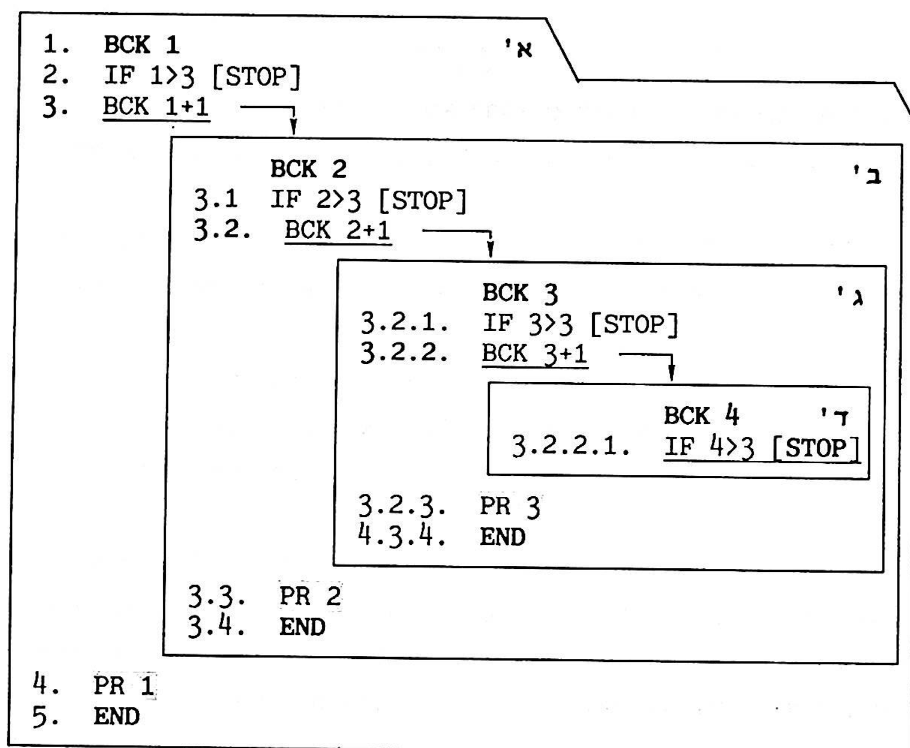
כאשר מחזור זה מסתיים, מסתיימת ההוראה 4.3.3, והמחשב עובר להוראה שלאחריה, אל ההוראה 4.3.4. כאן מופיעה המילה **END** האומרת שאין יותר הוראות לתיאור התהליך, וכך מסתיים מחזור ג'.

סיום מחזור ג' פירושו גמר ביצוע ההוראה 4.3, ועתה עובר המחשב להוראה 4.4. כאן המילה **END** מסיימת מחזור ב', שפירושו סיום ההוראה 4. ועם סיום ההוראה 4, מבצע המחשב את ההוראה 5. הפעם מציינת המילה **END** את סיום מחזור א'. וכך, עם גמר ביצוע מחזור א', מגיע תהליך ביצוע ההליך הרקורסיבי לסיומו.

בדוגמה הבאה נבחן את מהלך הביצוע של ההליך BCK. הליך זה דומה ל-FORWD, אלא **שהוראת הקריאה** הינה **לפני הוראת ההדפסה PR**.

```
TO BCK :MIS
IF :MIS > 3 [STOP]
BCK :MIS + 1
PR :MIS
END
```

כאשר נרשום BCK 1, הפלט שיופיע על המסך יהיה שונה מזה שקיבלנו על-ידי הרצת FORWD. העזר בתרשים הבא, ועקוב אחר ההוראות המתבצעות בזו אחר זו, על מנת שתוכל לרשום את הפלט באופן ידני, לפני שתריץ את ההליך במחשב.



שים לב, שהמחשב לא עובר להוראה 4 ('PR 1') של המחזור הראשון, כל עוד לא סיים את ביצוע **כל** ההוראות הכלולות במחזור שלאחריו.

כן הדבר לגבי הוראה 3.3 ('PR 2'), שבמחזור ב'. הוראה זו תתבצע רק בתום ביצוע ההוראה 3.2, המציינת את גמר ביצוע מחזור ג'.

ההוראה 3.2.2 שבמחזור ג' פונה למחזור ד', אך זה נעצר על-ידי ההוראה **STOP**, בשל קיומו של התנאי.

כאשר מחזור ד' מסתיים, הביצוע עובר להוראה 3.2.3 הנמצאת אחרי הקריאה, ולכן מודפס **המספר** 3. עם סיום מחזור ג' על-ידי **END**, עובר המחשב להוראה 3.3, המדפיסה את **המספר** 2, וכאן מסתיים מחזור

ב' - אף הוא על-ידי **END**. הביצוע מתקדם להוראה 4, אשר מדפיסה את **המספר 1**, וממשיכה למילה **END** של מחזור א' המציינת את סיום הביצוע של כל ההליך.

מהלך הביצוע של פונקציה רקורסיבית

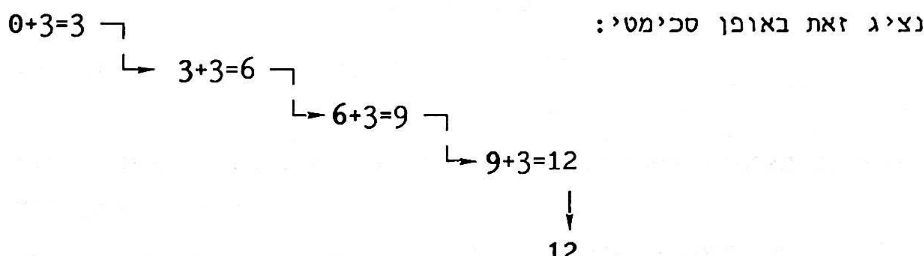
נלמד עתה את סדר ביצוע הפעולות בפונקציה רקורסיבית.

נגדיר פונקציה MULT, המקבלת 2 נתונים a ו-b, ומחזירה את מכפלתם (a * b). נניח שפעולת הכפל '*' אינה מוגדרת בגירסת לוגו שבידך, ואף לא הפונקציה PRODUCT.

אם יש לחשב לדוגמה 3*4, כלומר 4 פעמים 3, מהלך החישוב הוא:

((((0+3)+3)+3)+3)

נציג זאת באופן סכימטי:



לצורך הפתרון, נשתמש במשתנים :A ו-:B ונגדיר משתנה נוסף :RESULT, אשר ישמש כמשתנה צובר של התוצאה. בתחילה ערכו הוא '0', לאחר מכן מתוסף לו בכל מחזור הערך 3, וסה"כ 4 פעמים.

הגדרת MULT

```
TO MULT :A :B :RESULT
IF :B < 1 [OP :RESULT]
OP MULT :A (:B - 1) (:RESULT + :A)
END
```

כאשר נרשום ---> PR MULT 3 4 0

נקבל: 12

הפונקציה MULT מחשבת את הערך של 'A: פעמים B:'. B: משמש כמשתנה מונה בשיטת הספירה לאחור, כך שבכל פעם מופחת ממנו 1.
למשתנה RESULT: כאמור, מתוסף בכל מחזור הערך של A:. ולכן בקריאה לרקורסיה אנו רושמים:

```
OP MULT :A (:B - 1) (:RESULT + :A)
```

במחזור n, שהוא המחזור האחרון, מתקיים התנאי 'IF :B<1'. אז הוא מוסר את הערך של RESULT: למחזור שקדם לו, כלומר לזה שפנה אליו:

```
IF :B < 1 [OP :RESULT]
```

הפלט של מחזור n הופך להיות התוצאה של המחזור המקבל (מחזור n-1). ערך זה עליו להחזיר למחזור שפנה אליו מחזור n-2, וכן הלאה עד למחזור הראשון, שאליו אנו פונים כאשר רושמים

```
PR MULT 3 4 0
```

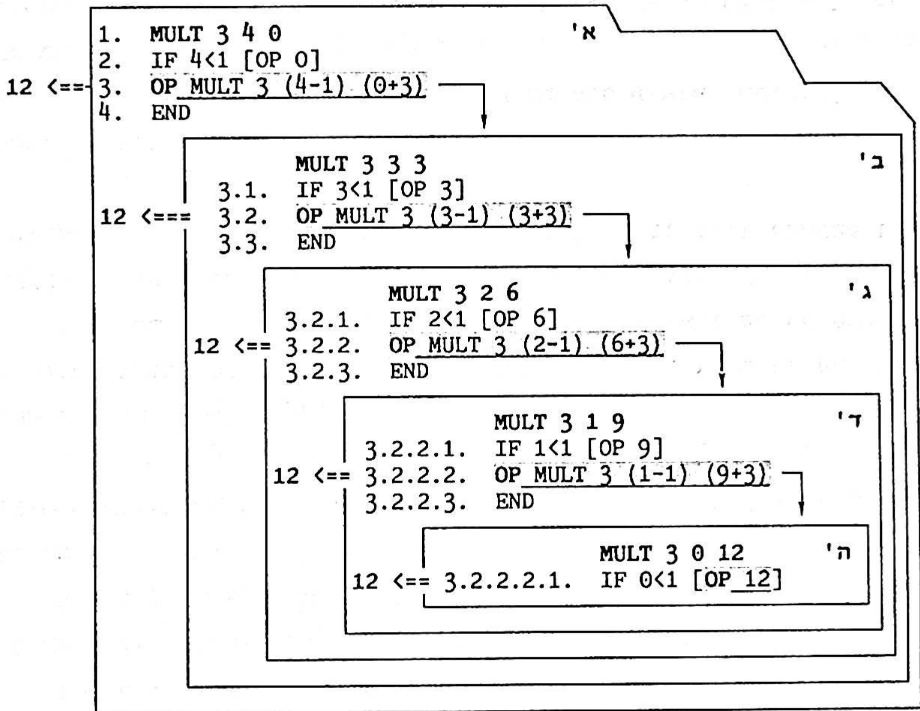
עתה התוצאה מודפסת על המסך.

אנו רואים, שבתהליך זה הפלט של הפונקציה הנקראת משמש קלט לפונקציה הקוראת.

נמחיש זאת באמצעות התרשים הבא.

כאשר נרשום --> ?PR MULT 3 4 0

יתחיל הביצוע:



בהוראה 2 של המחזור הראשון, אין התנאי מתקיים, לכן עובר המחשב להוראה 3. על הוראה זו לפלוט (Output) את **ערך התוצאה** של מחזור ב', והביצוע עובר למחזור חדש.

גם במחזור ב' אין התנאי מתקיים והביצוע עובר להוראה 3.2. הפלט של מחזור זה תלוי אם כן, **בתוצאת החישוב** של מחזור ג'.

כתוצאה מאי קיום התנאי אף במחזור ג', הפקודה OP שבהוראה 3.2.2 תפעל על **הערך שיחושב** על-ידי מחזור ד', אשר משמש עבורה כקלט. במחזור ד' עדיין התנאי לא מתקיים, ולכן ההוראה 3.2.2.2 תפלוט את **הערך שיתקבל** ממחזור ה'.

במחזור ה', כאשר התנאי כן מתקיים, מחזירה OP שבמשפט התנאי את

הערך 12 (הוראה 3.2.2.2.1). ערך זה מהווה קלט עבור מחזור ד', אותו הוא מוסר למחזור ג'. הפקודה OP בהוראה 3.2.2 תחזיר ערך זה למחזור ב', אשר יפעל עליו בהוראה 3.2 וימסור אותו למחזור א'. הוראה 3 של מחזור א' מקבלת תוצאה זו ומחזירה אותה כערך סופי של כל החישוב.
שים לב, כי המילה END בהגדרה מציינת את סוף **תיאור** החישוב, או התכנית, בלבד.

התוצאה המחושבת על-ידי פונקציה הינה **ערך**, ולכן ניתן להשתמש בה כחלק של ביטוי חשבוני. כמו כן למדנו ש-OP פועלת על ביטויים הניתנים לחישוב. לכן נוכל להשתמש בתוצאת החישוב של כל מחזור ומחזור במקום המשתנה RESULT:, ולתוצאה זו נוסיף את הערך של A: במשפט הפלט 'OP'.

נניח שהתוצאה של מחזור ב' ידועה, אזי נוכל לחשב את מחזור א' על פי התבנית הבאה:

מחזור א' = A: + מחזור ב'

כן הדבר לגבי מחזור ב'. אם התוצאה של ג' ידועה אזי:

מחזור ב' = A: + מחזור ג'

ובאופן כללי:

מחזור n = A: + מחזור n+1

החישובים מתבצעים על ערכים לא ידועים, התלויים בתוצאת חישובים אחרים, שאף הם לא ידועים. על כן, הדבר מחייב לחשב ערך התחלתי ידוע, אשר ישמש **נקודת מוצא** לחישוב של כל פונקציה. ואמנם, תוצאת הכפל של נתון כלשהו ב-0 שווה 0, **לכן** '0' יהיה ערך הפלט של המחזור האחרון, מחזור n. הפלט של המחזור האחרון הינו **הערך הראשון** המוחזר כערך התוצאה של המחזור אשר קרא לו. **ערך התוצאה הראשון** הנמסר על-ידי מחזור n למחזור הקורא לו n-1, הינו הערך המשמש **נקודת מוצא** לחישוב ערך הפונקציה כולה. התוצאה של המחזור האחרון נמסרת רק כאשר ערך המכפיל B: שווה 0 (אפס), כלומר:

```
IF :B < 1 [OP 0]
```

את התוצאה הסופית מקבלים על-ידי הצבה של הערך המתקבל מן המחזור הנקרא אל הביטוי המחשב את ערך הפלט של המחזור הקורא.

למעשה, הפתרון הנכון מסתמך על שתי המשוואות הבאות:

- עבור המחזור האחרון, התוצאה היא 0
- עבור כל מחזור ת אחר, התוצאה היא: :A + מחזור n+1

נבחן עתה את מהלך הביצוע של ההליך MULTPL:

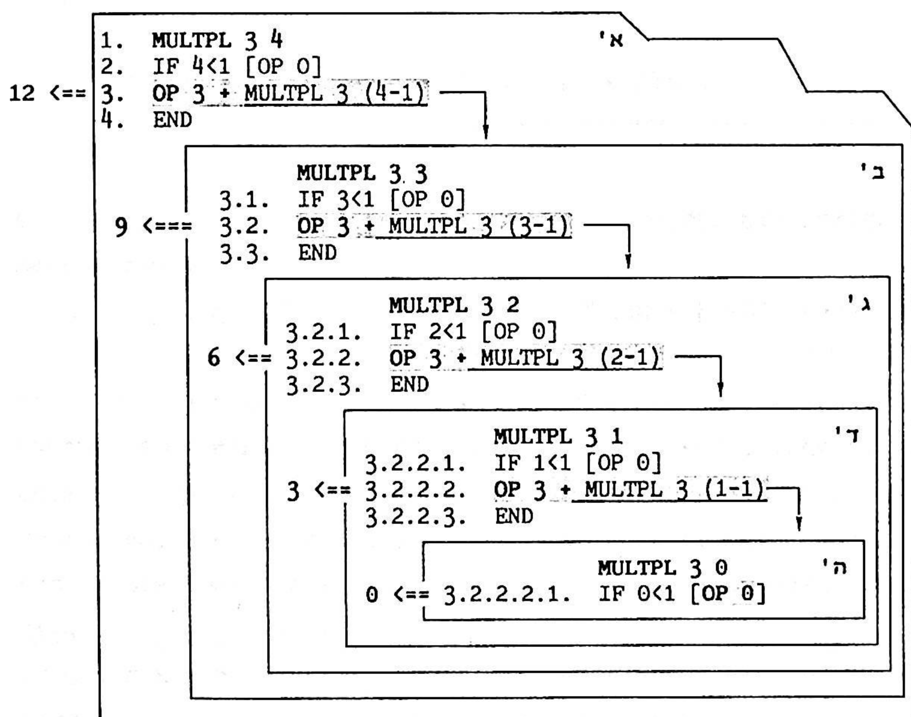
הגדרת MULTPL

```
TO MULTPL :A :B
IF :B < 1 [OP 0]
OP :A + MULTPL :A (:B - 1)
END
```

נסה להבין את תהליך ביצוע ההוראות המובא בתרשים שלפניך.

כאשר נרשם <--- ?PR MULTPL 3 4

יתחיל הביצוע:



שים לב, כי בדרך חישוב זו תוצאת הפונקציה במחזור **n-1** שווה לתוצאת הפונקציה של מחזור **n בתוספת הערך של A**:. כלומר, תוצאת החישוב של כל מחזור תלויה בתוצאת החישוב של המחזור הבא אחריו, אך מחושב לפניו. כך הדבר בכל מחזור עד למחזור האחרון, שבו התנאי אמנם מתקיים, ואז התוצאה שלו משמשת נקודת מוצא התחלתית של החישוב כולו. בתום הביצוע מחזירה הפונקציה את ערך התוצאה הסופית.

במחזור האחרון, מחזור ה', התוצאה שלו מהווה נקודת מוצא:

```
0 <<===  'OP 0'
```

ערך התוצאה של מחזור ד' שווה ל-(מחזור ה' + 3):

```
3 <<===  'OP 3 + MULTPL 3 (1 - 1)' = 3 + 0
```

ערך הפונקציה של מחזור ג' מבוטא על-ידי (מחזור ד' + 3):

```
6 <<===  'OP 3 + MULTPL 3 (2 - 1)' = 3 + 3
```

תוצאת החישוב של מחזור ב' היא (מחזור ג' + 3):

```
9 <<===  'OP 3 + MULTPL 3 (3 - 1)' = 3 + 6
```

ערך התוצאה של מחזור א' שהוא הערך הסופי של הפונקציה הרקורסיבית שווה ל-(מחזור ב' + 3):

```
12 <<===  'OP 3 + MULTPL 3 (4 - 1)' = 3 + 9
```

במהלך הביצוע הרקורסיבי נעשית **קריאה רקורסיבית** ממחזור למחזור. **הקריאה הרקורסיבית הראשונה** מתבצעת במחזור הראשון, ובכל מחזור מתבצעת קריאה נוספת עד למחזור האחרון, שם היא לא מתבצעת כלל. במקום זאת, מוחזר ערך המשמש כערך התחלתי לחישוב. כלומר, המחזור האחרון הוא המחזור הראשון, אשר מחשב ומעביר ערך למחזור שקרא לו.

כל מחזור אשר מקבל ערך מבצע חישוב על הערך שנמסר לו, ומעביר את תוצאת החישוב שלו למחזור שקדם לו, עד למחזור הראשון. המחזור הראשון פולט את התוצאה הסופית של כל חישוב.

סידרת פיבונצ'י

נגדיר פונקציה המחשבת את האיבר ה-n-י בסידרת פיבונצ'י:

1,1,2,3,5,8,13,21,34,...

שים לב שגם האיבר הראשון וגם השני שווים ל-1 ($a_1=1$,$a_2=1$).
כל איבר אחר שווה **לסכום** של שני קודמיו:

$a_k = a_{k-1} + a_{k-2}$.

הגדרת FIB

```
TO FIB :N
IF :N = 1 [OP 1]
IF :N = 2 [OP 1]
OP (FIB :N-1) + (FIB :N-2)
END
```

מאחר שתוצאת כל מחזור תלויה בשני מחזורים הנקראים על-ידו, יש להגדיר שני ערכים התחלתיים, אשר ישמשו כצעד ראשון לחישוב.

החישוב עבור **FIB 6**, למשל:

```
FIB 6 = FIB 5 + FIB 4
        FIB 5 = FIB 4 + FIB 3
                FIB 4 = FIB 3 + FIB 2
                        FIB 3 = FIB 2 + FIB 1
```

נתון: **FIB 2 = FIB 1 = 1**

נציב ערך זה בביטוי החישוב של FIB 3 ונקבל:

FIB 3 = 1 + 1 = 2

אם נציב את הערכים המתאימים
עבור ביטוי החישוב של FIB 4, נקבל:

FIB 4 = 2 + 1 = 3

כך נעשה עבור החישוב של FIB 5, ונקבל: FIB 5 = 3 + 2 = 5

ותוצאת החישוב עבור FIB 6 תהיה: FIB 6 = 5 + 3 = 8

ואכן, האיבר השישי הוא 8.

* * * * *

תרגילים (המשך)

16. הגדר פונקציה POWERN, המקבלת שני מספרים טבעיים, ומחשבת את הנתון A: בחזקת X:. זכור ש-A: בחזקת 0 שווה ל-1.
אם הינך משתמש בגירסה של IBM LOGO, הנח שלא קיימת הפונקציה POWER.

17. הנח שפעולות הכפל והחילוק אינן מוכרות על-ידי המחשב, ואף לא הפונקציות REMAINDER ו-QUOTIENT, בתנאים אלה הגדר פונקציה MANA, המקבלת שני נתונים a ו-b, ומחשבת את המנה של חליקת a ב-b.
הוראות החישוב:
א. אם a<b אזי התוצאה היא '0'.
ב. יש למנות את מספר המחזורים בהם ניתן לחסר את b מ-a
(OP 1 + MANA (:A - :B) :B)

18. הסתמך על אותה הנחה שבתרגיל 17 על-מנת להגדיר פונקציה SHERIT, לחישוב השארית של חלוקת a ב-b.

19. הגדר פונקציה FACTORIAL, לחישוב N! (N עצרת).
נוסחת החישוב של N עצרת:
N! = N*(N-1)*(N-2)* ... *3*2*1
לדוגמה - 5! = 5*4*3*2*1

על פי ההגדרה N! = N * (N - 1)!
וגם 1! = 1

למעשה, אם !4 ידוע, אז: 5! = 5 * 4!

כמו כן, אם !3 ידוע, אז: 4! = 4 * 3!

וכן הלאה ...

<u>הנחיה</u>: השתמש בהגדרה '1 = !1' כנקודת מוצא התחלתית לחישוב.

20. הגדר פונקציה SUMFIB, המשתמשת ב-FIB, לחישוב **סכום** של טור מסידרת פיבונצ'י, בעל N: איברים.

<u>הדרכה:</u>

- עבור טור בעל איבר אחד, כתוב: SUMFIB 1 = FIB 1
- סכום טור בעל N: איברים שווה לסכום טור של N-1: איברים, ועוד ערך האיבר ה-N:, מכאן ש-

SUMFIB :N = SUMFIB (:N-1) + FIB :N

<u>הערה</u>: יש לציין, שלרוב התרגילים יש לפחות שני פתרונות שונים.

21. הגדר פונקציה רקורסיבית לחישוב סכום של N: האיברים הראשונים לכל אחת מן הסדרות הבאות:

א. 1 + 1/2 + 1/3 + ... + 1/:N

ב. 1 + 1/2 - 1/3 + 1/4 - 1/5 + ... ± 1/:N

ג. ((... ((1+2)*3+4)*+ ... +:N-1)*:N)

פרק 10

ביטויים בוליאנים

השימוש במחשבים אינו מצטמצם בביצוע חישובים חשבוניים בלבד, אלא כולל גם חישובים לוגיים. עד כה ביצענו חישובים לוגיים בביטויים המשמשים להשוואת יחסים של ערכים מספריים. זאת עשינו באמצעות **פעולות היחס**, גדול מ- ('<'), קטן מ- ('>') ושווה ל- ('=').

ביטוי לוגי ופעולות לוגיות

ביטוי אשר משתמשים בו בפעולות היחס נקרא **ביטוי לוגי פשוט**. הוא מופיע במשפטי התנאי אשר בודקים אם הביטוי הלוגי הוא **אמיתי** או **שקרי**, כלומר "נכון" (TRUE) או "לא נכון" (FALSE). מידע זה דרוש על מנת לבצע פעולה מסוימת במקרה אחד, עם אפשרות לפעולה אחרת במקרה אחר.

לדוגמה, כאשר הגדרנו פונקציה ABS, ביקשנו מהמחשב לבדוק אם הנתון X: קטן מ-0: 'IF :X < 0'. הערך של ביטוי זה יכול להיות 'TRUE' או 'FALSE' בלבד. בהתאם לערך שמתקבל עוברת התכנית להוראה שיש לבצע.

לפעמים יש צורך לבדוק יותר מאשר ביטוי לוגי פשוט אחד בתוך הוראת תנאי אחד. בלוגו, כמו בשפות אחרות, ניתן לכתוב ביטוי לוגי המורכב ממספר ביטויים לוגיים פשוטים. הקשר המחבר ביטויים לוגיים פשוטים לביטוי לוגי מורכב נעשה על-ידי פעולות הנקראות **מילות חיבור** מבחינה דקדוקית, **וקשרים** מבחינה לוגית. הבדיקה של הוראת התנאי תתייחס **לביטוי הלוגי המורכב כולו**. כלומר, ניתן להגיע להחלטות בכל רמת מורכבות באמצעות משפטי התנאי. ערך

הביטוי המורכב יהיה תלוי אם כן:-

א. בערכם של הביטויים הפשוטים.

ב. בפעולה הלוגית.

מבחינים בין קשרים חד-מקומיים ודו-מקומיים. הפעולה הלוגית 'NOT' היא קשר חד-מקומי, הפעולות הלוגיות 'AND' ו-'OR' מייצגות קשרים דו-מקומיים. פעולות לוגיות המתבצעות על ביטויים שערכם הוא 'אמת' או 'שקר', נקראות גם **פעולות בוליאניות**, על שמו של המדען הצרפתי ג'ורג' בול (George Bool). הוא חקר אותן ביסודיות והניח את הבסיס התיאורטי לפעולות אלו. בכך הוא איפשר חישובים בביטויים בוליאנים, דהיינו ביטויים לוגיים מורכבים.

הפעולה הלוגית AND, פירושה **'וגם'**. בפעולה זו אנו קושרים שני ביטויים לוגיים פשוטים, אשר נכנה אותם Logi1 ו-Logi2, לביטוי לוגי מורכב אחד, שנכנה אותו Bool:

Bool = Logi1 AND Logi2

Bool יקבל ערך 'אמת' רק כאשר **גם** Logi1 **וגם** Logi2 יחד מייצגים ערכי 'אמת'. די באחד מהם שיש לו ערך 'שקר' על מנת שהביטוי Bool ייחשב 'שקרי'.

הפעולה הלוגית OR, פירושה **'או'**. ערכו של הביטוי המורכב:

Bool = Logi1 OR Logi2

הביטוי Bool יקבל ערך 'אמת', אם ל-Logi1 ערך 'אמת' **או** ל-Logi2 ערך 'אמת'. במילים אחרות, די באחד מהם שיהיה לו ערך 'אמת' כדי שהביטוי Bool יהיה 'אמת'.

הפעולה הלוגית NOT, פירושה **'לא'**, והיא פועלת על ביטוי לוגי פשוט אחד בלבד. אם נתון הביטוי Logi1, הערך המתקבל מהביטוי המורכב 'NOT Logi1', יהיה תמיד ההיפך מן הערך של הביטוי Logi1. בדרך זו NOT פועלת למעשה לשלילת הביטוי הלוגי.

לכל שפת תכנות יש מבנה תחבירי המיוחד לה. בלוגו נמצא, ששלוש הפעולות AND, OR ו-NOT, הן פעולות מסוג **Prefix**. כלומר, פעולות המקדימות את הביטויים שיש לפעול עליהם. בשפות אחרות נמצא, שהפעולות AND ו-OR הן מסוג **Infix**. כלומר, פעולות המופיעות בין שני הביטויים (כפי שהובא לעיל).

הפעולות האריתמטיות למשל, הינן מסוג Infix, כאשר הן נמצאות בין שני ביטויים חשבוניים, לדוגמה: '540 + 773', '36 * 15'.
הערה: בלוגו פעולות אלו מתנהגות גם כ-Prefix.

לעומת זאת, SUM ו-PRODUCT הן פעולות מסוג Prefix, מכיון שהן מופיעות לפני הביטויים: 'SUM 540 773', 'PRODUCT 36 15'.

המבנה הכללי של ביטוי לוגי המורכב באמצעות AND:

AND <ביטוי לוגי 1> <ביטוי לוגי 2>

לפי ערכי שני הביטויים הלוגיים, נמצא 4 תוצאות:

ערך הביטוי	תוצאה
AND "TRUE "TRUE	TRUE
AND "TRUE "FALSE	FALSE
AND "FALSE "TRUE	FALSE
AND "FALSE "FALSE	FALSE

המבנה הכללי של ביטוי המורכב באמצעות OR:

<ביטוי לוגי 2> <ביטוי לוגי 1> OR

נציג את 4 האפשרויות של ערכי הביטויים הלוגיים ונקבל את התוצאה:

ערך הביטוי	תוצאה
OR "TRUE "TRUE	TRUE
OR "TRUE "FALSE	TRUE
OR "FALSE "TRUE	TRUE
OR "FALSE "FALSE	FALSE

המבנה הכללי של ביטוי NOT:

<ביטוי לוגי> NOT

עבור ביטוי לוגי אחד יש רק שתי אפשרויות:

ערך הביטוי	תוצאה
NOT "TRUE	FALSE
NOT "FALSE	TRUE

<u>שים לב</u>, שהפעולות הבוליאניות הן למעשה **פונקציות בוליאניות**. הן דורשות קלט עם ערכים של 'TRUE' או 'FALSE', ומחזירות כפלט ערך 'אחד בלבד, אף הוא 'TRUE או 'FALSE'.

באמצעות הפעולות AND ו-OR ניתן לפעול על יותר מאשר שני ביטויים, אם נתחום את כל הביטוי המורכב בסוגריים עגולים:

(AND ביטוי לוגי 1 ביטוי לוגי 2 ... ביטוי לוגי n)
(OR ביטוי לוגי 1 ביטוי לוגי 2 ... ביטוי לוגי n)

בפעולות AND די בביטוי לוגי אחד שקרי, כדי שערכו של כל הביטוי המורכב יהיה 'שקר'.

בפעולות OR, די בביטוי אמיתי אחד, על מנת שערכו של כל הביטוי המורכב יהיה 'אמת'.

אם נרשום ---> ?PR (AND 1=1 (SQRT 25)=5 12>11 1=0 4=2*2)
נקבל על המסך את ההודעה: FALSE

הביטוי '1=0' הוא FALSE, ולכן קיבלנו שערך הביטוי כולו הוא FALSE. ההוראה PR פועלת על הערך המתקבל, ולכן היא מדפיסה את המילה 'FALSE'.

תכנית לבדיקת תקינות של תאריך

נגדיר הליך המקבל תאריך המורכב משלושה נתונים - שנה, חודש ויום, ומבצע עליו בדיקת תקינות עבור שנת 1988.

הגדרת DATE

```
TO DATE :YEAR :MONTH :DAY
IF NOT :YEAR = 1988 [PR [INVALID YEAR] STOP]
IF OR :MOMTH < 1 :MONTH > 12 [PR [INVALID MONTH] STOP]
IF AND :DAY > 29 :MONTH = 2 [PR [INCORRECT DATE] STOP]
IF (OR :MONTH = 4 :MONTH = 6 :MONTH = 9 :MONTH = 11 )          !
                [IF :DAY > 30 [PR [INCORRECT DATE] STOP]
IF AND :DAY>0 :DAY<32 [PR [IT'S CORRECT] [PR [INVALID DAY]
END
```

<u>הסבר</u>:

א. אם הערך של המשתנה YEAR: **שונה** מ-1988, הדפס הודעת שגיאה ועצור. **שונה**, כלומר, **לא שווה**:

```
NOT :YEAR = 1988
```

ב. אם ערך MONTH: אינו בתחום המספרים (1...12), הדפס הודעת שגיאה ועצור. אם הערך קטן מ-1 **או** גדול מ-12, המספר אינו בתחום המספרים המצביעים על חודש מסוים:

```
OR :MONTH < 1 :MONTH > 12
```

ג. בחודש פברואר של שנת 1988 יש 29 יום. לכן יש להדפיס הודעת שגיאה ולעצור, אם ערכו של DAY: גדול יותר מ-29 **וגם** ערכו של MONTH: הוא 2:

```
AND :DAY > 29 :MONTH = 2
```

ד. אם ערכו של MONTH: מצביע על החודש ה-4, **או** ה-6, **או** ה-9, **או** ה-11, בדוק אם ל-DAY: ערך גדול מ-30 ואם כן, הדפס הודעת שגיאה ועצור:

```
(OR :MONTH=4 :MONTH=6 :MONTH=9 :MONTH=11)
```

ה. אם ערכו של DAY: הוא בתחום המספרים (1...31), המצביע על יום סביר בחודש, הדפס שהתאריך נכון. אחרת, הדפס הודעת שגיאה:

```
AND :DAY > 0 :DAY < 32
```

פעולות אלו מתבצעות על ביטויים עם ערכים של 'TRUE' או 'FALSE', ולכן הן פועלות על כל ביטוי לוגי שהוא, בין אם הוא פשוט או מורכב. ננצל עובדה זו ונגדיר מחדש את ההליך הזה. בהוראות התנאי יופיעו ביטויים ברמת מורכבות גבוהה יותר, ועל-ידי כך נצמצם את מספר הוראות התנאי.

הגדרת DATE1

```
TO DATE1 :Y :M :D
IF (OR NOT :Y=1988 :M<1 :M>12 :D<1 :D>31)                          !
                        [PR [INVALID DATE] STOP]
IF OR AND :M=2 :D>29 AND :D>30 (OR :M=4 :M=6 :M=9 :M=11)           !
          [PR [INCORRECT DATE]] [PR [IT IS CORRECT]]
END
```

ניתן להלן הסבר על אופן כתיבת הביטוי מבחינה לוגית, ונלמד תוך כדי כך גם על הדרך בה פועלות פונקציות אלו בלוגו.

בהוראת התנאי הראשונה - אם התנאי מתקיים, כלומר אם ערכו של הביטוי 'אמת', מודפסת הודעת שגיאה. אנו יודעים שהתאריך איננו תקין, אם לאחד מן הביטויים הבאים יש ערך 'אמת':

א) NOT :Y = 1988
ב) :M < 1
ג) :M > 12
ד) :D < 1
ה) :D > 31

כלומר, מספיק שאחד מן הביטויים האלה יהיה בעל ערך 'אמת', על מנת שנדפיס הודעת שגיאה. לכן, קשרנו את כל הביטויים האלה באמצעות הפעולה OR.

יש להבחין בכך שהביטוי הראשון 'NOT :Y = 88', הוא ביטוי מורכב, והוא אחד הקלטים של הפונקציה OR. על כן מחשבים אותו תחילה, כדי שפעולת OR תפעל על הערך המתקבל מחישוב זה.

בהוראת התנאי השניה - אם התנאי מתקיים, מודפסת הודעת שגיאה, אחרת - מודפסת הודעה חיובית. כאן מתבצעת הבדיקה עבור חודש פברואר, ועבור החודשים שיש בהם רק 30 יום.
התאריך לא יהיה תקין:

- אם ל-D: יהיה ערך גדול מ-29 עבור חודש 2 (פברואר):

AND :M=2 :D>29

- **או**, אם נתון ל-D: ערך גדול יותר מ-30, עבור אחד מהחודשים 9, 4, 6 ו-11:

AND :D>30 (OR :M=4 :M=6 :M=9 :M=11)

תוצאת החישוב של כל אחד משני ביטויים אלה, מהווה קלט עבור הפעולה OR הבאה בתחילת הביטוי המורכב כולו. כלומר, על המחשב לחשב תחילה את הערך של כל אחד מהם.

בדוגמה זו, הקלט השני הינו ביטוי מורכב, המחושב על-ידי הפונקציה AND. כאן פועלת AND על שני ערכים המתקבלים משני קלטים, שאחד מהם אף הוא ביטוי מורכב:
(OR :M=4 :M=6 :M=9 :M=11)

בחישוב של ביטוי לוגי מורכב מבצעים תחילה את החישוב של כל אחד מהביטויים המשמשים כקלט, ורק לבסוף ניתן לחשב את הערך של הביטוי השלם.

חישוב פונקציונלי

הפונקציות הבוליאניות בלוגו הן מסוג Prefix, אשר בהן הקלטים נמצאים בחלק הימני של הביטוי הבוליאני. אם קלטים אלה מופיעים אף הם כביטויים בוליאנים, יהיו גם הקלטים שלהם בחלק הימני בביטוי המשמש כקלט, וכן הלאה. מכיון שהחישוב מתבצע תחילה על הביטויים המשמשים כקלט, מתברר שהחישוב מתבצע מימין לשמאל.

באריתמטיקה קיימת דרך חישוב מסוימת הנקראת **כתיב פולני** (Polish Notation), שבה מופיעה פעולת החשבון לפני שני הנתונים ולא ביניהם.

לדוגמה: - הביטוי '8 * 54' נכתב כך: 8 54 *
- הביטוי '2 - 5 * 40' נכתב כך: 2 5 40 * -

שים לב, כי פעולת החיסור נמצאת לפני שני הנתונים שהיא פועלת עליהם: האחד הוא הערך המתקבל מחישוב הביטוי '5 40 *', והשני הוא 2.

ביטוי המתבצע מימין לשמאל, כאשר הפעולה נמצאת בתחילת הביטוי, דומה לביטוי הפולני שהחישוב בו מתנהג **באופן פונקציונלי**. נמצא שבביטוי מורכב, פונקציה אחת פועלת על פונקציה שניה, אשר פועלת על פונקציה שלישית וכו'. כך שהפלט של הפונקציה השלישית משמש קלט לפונקציה השניה, והתוצאה של השניה משמשת קלט לפונקציה הראשונה. במילים אחרות - ערך הפונקציה השלישית משמש ארגומנט לפונקציה השניה, וערך הפונקציה השניה מהווה ארגומנט לפונקציה הראשונה.

לדוגמה: נתונות שלוש פונקציות F(X), G(Y), H(Z), המוגדרות באופן הבא:

F(X) = X+2

G(Y) = Y*2

H(Z) = Z*Z

כאשר נרצה לחשב את הביטוי F(G(H(3))), נבצע את הצעדים הבאים:

תחילה נחשב H(3) ונקבל ===> 9

לאחר מכן נחשב G(9) ===> 18

ולבסוף נחשב F(18) ===> 20 וזוהי התוצאה.

פרדיקטים

בלוגו קיימות מספר פונקציות, הבודקות תנאים מסוימים. פונקציה כזו קרויה **פרדיקט** (Predicate). ערכיה האפשריים הם "TRUE או "FALSE".

לדוגמה, הפרדיקט **NUMBERP** מקבל נתון קלט אחד, ומחזיר את הערך 'TRUE', רק אם הקלט הוא בעל ערך מספרי.

כאשר נרשום ---> (הערך המספרי 3.14) ?PR NUMBERP PI

נקבל: TRUE

אבל

כאשר נרשום ---> (המילה PI) ?PR NUMBERP "PI

נקבל תשובה שלילית: FALSE

בתכניות שבהן על המשתמש להקיש נתון מספרי כלשהו, ניתן באמצעות NUMBERP לבצע בדיקה של הקלט המוקש ולהמשיך את ביצוע התכנית בהתאם לתוצאה שתתקבל. כך נוכל להמנע מהודעות שגיאה ועצירת התכנית בעת הביצוע.

פרדיקט אחר הוא **EQUALP**, הבודק שוויון בין שני נתונים. הוא פועל על נתונים מספריים וגם על נתונים לא מספריים, בדומה לפעולה '='.

כאשר נרשום ---> ?PR EQUALP 4*9 6*6

נקבל: TRUE

כאשר נרשום ---> ?PR EQUALP 4+9 6+6

נקבל: FALSE

נרשום דוגמה נוספת ---> ?PR EQUALP "ABC "ABC

המילים שוות והתוצאה: TRUE

עתה נרשום ---> ?PR EQUALP "ISRAEL [ISRAEL]

אין שיוויון מכיון שמשווים מילה עם רשימה: FALSE

יוצרי השפה דאגו ששמות הפרדיקטים בלוגו יסתיימו באות 'P', כדי לציין שמדובר בפרדיקט המחזיר פלט 'אמת' או 'שקר'.
גם אנו נוכל להגדיר פונקציות לוגיות לביצוע בדיקות מסוימות.

נגדיר להלן פונקציה המקבלת מספר אחד ופולטת את המילה 'TRUE' אם הערך הוא שלילי, אחרת התוצאה תהיה 'FALSE'.

הגדרת SHLILIP

```
TO SHLILIP :MIS
IF :MIS < 0 [OP "TRUE] [OP "FALSE]
END
```

ידוע שהמחשב מבצע חישוב על ביטויים לוגיים, כמו כן ידוע ש-OP פועלת על ביטויים הניתנים לחישוב באמצעות המחשב. ולכן נוכל להגדיר פונקציה זו בדרך הפשוטה הבאה:

הגדרת NEGATIVEP

```
TO NEGATIVEP :NUM
OP :NUM < 0
END
```

הביטוי ':NUM < 0' הוא ביטוי לוגי ולכן הערך שלו הוא 'TRUE' או 'FALSE'. הפעולה OP תפיק במקרה זה רק אחד משני ערכים אלה, והרי לפנינו פונקציה לוגית.

* * * * *

תרגילים

1. כתוב פונקציה לוגית, המקבלת 4 נתונים, בודקת אם **כולם** שווים, ופולטת **הודעה** מתאימה.

הערה: OP פולטת את ערך הפונקציה, אשר יכול להיות מספר, מילה או רשימה. אם נרצה שערך הפונקציה יהיה הודעה, נוכל לרשום את ההודעה ברשימה, למשל:

OP [YES, IT IS TRUE]

2. כתוב פרדיקט המקבל 3 נתונים, ובודק אם הם יכולים להוות צלעות של משולש. הפלט הוא 'TRUE' או 'FALSE'.

3. כתוב פונקציה המקבלת 3 נתונים, בודקת אותם ומפיקה תשובה בהתאם לתוצאת הבדיקה:

'0' אם אי אפשר לבנות בהם משולש.
'1' אם אפשר ליצור משולש שווה צלעות.
'2' אם אפשר ליצור משולש שווה שוקיים.
'3' אם המשולש המתקבל יהיה שונה צלעות.

אפשר להשתמש בפרדיקט שבתרגיל 2 לפתרון תרגיל זה..

4. באחד התרגילים בשיעורים הקודמים הגדרת תכנית המקבלת 4 נתונים, ובודקת אם ניתן ליצור בהם מרובע, אשר אפשר יהיה לחסום בו עיגול. פתור מחדש תרגיל זה, אך הפעם השתמש בפעולות הלוגיות, והגדר את התכנית כפונקציה.

5. הגדר פרדיקט ODDP לבדיקה אם נתון הוא אי-זוגי. הגדר פרדיקט אחר EVENP לבדיקה אם נתון הוא זוגי. רצוי שתבדוק תחילה אם הנתון הוא ערך מספרי.

6. הגדר פונקציה SYMMETRYP המקבלת מספר NUM: המורכב מחמש ספרות, בודקת אם הספרות שלו הן סימטריות (הסיפרה הראשונה זהה לאחרונה, והשניה זהה לרביעית), ופולטת הודעה בהתאם.

7. הגדר פונקציה רקורסיבית MIUN3, למיון שלושה מספרים שונים בסדר עולה.

מאחר ש-OP פועלת על ערך אחד בלבד, יש לאחד את שלושת הערכים לרשימה אחת באמצעות ההוראה SE (רשימה אחת מהווה דבר אחד, כלומר, ערך אחד):

```
OP (SE :A :B :C)
```

כאשר נרשום למשל --->

```
?PR MIUN3 98 11 36
```

נקבל את הפלט הבא:

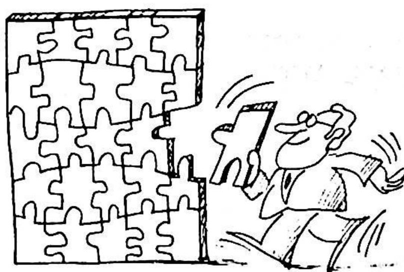
```
11 36 98
```

אם נרשום --->

```
?SHOW MIUN3 98 11 36
```

נקבל:

```
[11 36 98]
```



פרק 11

מילים ורשימות

עד כה השתמשנו **במשתנים** לצורך החישובים שביצענו באמצעות המחשב. ראינו שבהגדרת הליך אחד ניתן להשתמש במשתנים שונים, שכל אחד מהם מיועד לאחסנה של נתון אחר. כך למשל, המשתנה YEAR: שימש עבור השנה, MONTH: - עבור החודש והמשתנה DAY: - עבור היום. בכל רגע קשור המשתנה עם ערך אחד בלבד, כאשר קישור זה אינו קבוע.

דרך אחרת לאחסן נתונים הינה על-ידי סידור שלהם ברשימה המוצגת כמערך **חד-מימדי** (וקטור). באמצעות מערך כזה ניתן לטפל במספר נתונים כמו רשימת תלמידים למשל, רשימת מספרי טלפון או רשימת ארצות. דרך זו נוחה ופשוטה יותר במקום השימוש במספר רב של משתנים המיועדים לאחסן נתונים עבור חישובים זהים.

כבר בתחילת הספר הזכרנו את מושג **הרשימה** (List) בלוגו, כאשר למדנו את הוראות ההדפסה. שם נאמר שרשימה הינה אוסף של דברים התחום בין **סוגריים מרובעים**. כמו כן נאמר, שלוגו מתייחסת אל הרשימה כאל **דבר אחד**, גם אם היא מכילה מספר דברים.

אם נתונה הרשימה [34 21 13 8 5], אנו אומרים שהדברים המופיעים בה הם **איברי** רשימה זו. נוכל להצביע על כל אחד מאיברי הרשימה על פי מקומו הסידורי. איבר מסוים יכול להיות הראשון ברשימה, השני בה, או האחרון. ברשימה הנתונה, המספר '5' הוא האיבר הראשון, והמספר '34' הוא האיבר האחרון.

פונקציות בחירה

בלוגו קיימות פונקציות מיוחדות המשמשות לעיבוד רשימות. חלקן **פונקציות בחירה** (Selectors) המיועדות לבחור ולפלוט חלק מרשימה נתונה, או איבר אחד ממנה:

הפונקציה FIRST

כאשר הקלט הוא רשימה, הפלט הוא האיבר הראשון של רשימה זו.

כאשר נרשום ---> ?SHOW FIRST [5 8 13 21 34]

יודפס על המסך: 5

הפונקציה LAST

אם נתונה רשימה, הפלט הוא האיבר האחרון המופיע ברשימה זו.

כאשר נרשום ---> ?SHOW LAST [5 8 13 21 34]

נקבל על המסך: 34

הפונקציה BUTFIRST (בקיצור BF)

עבור קלט שהוא רשימה, הפלט הוא **רשימת** כל איברי הרשימה **פרט לראשון**.

כאשר נרשום ---> ?SHOW BF [5 8 13 21 34]

יופיע הפלט הבא: [8 13 21 34]

הפונקציה BUTLAST (בקיצור BL)

עבור קלט שהוא רשימה, הפלט הוא **רשימת** כל איברי הרשימה **פרט לאחרון.**

אם נרשום ---> ?PR BL [5 8 13 21 34]

נקבל את הפלט הבא: [5 8 13 21]

לפניך מוגדר הליך PRLIST המקבל רשימה כלשהי :VEC ומדפיס איברים שונים שלה, כמפורט להלן:

הגדרת PRLIST

```
TO PRLIST :VEC
PR :VEC              א. כל איברי הרשימה -
PR FIRST :VEC        ב. האיבר הראשון -
PR LAST :VEC         ג. האיבר האחרון -
PR BF :VEC           ד. רשימת כל האיברים פרט לראשון -
PR BL :VEC           ה. רשימת כל האיברים פרט לאחרון -
END
```

בקריאה להליך רושמים את שם ההליך ונותנים למשתנה :VEC קלט המוצג **כרשימה** ---> ?PRLIST [LONDON PARIS ROMA BERN]

כתוצאה מן הביצוע יופיע על המסך:

```
LONDON PARIS ROMA BERN
LONDON
BERN
PARIS ROMA BERN
LONDON PARIS ROMA
```

שים לב, כי השתמשנו בהוראה PR ולא ב-SHOW, ולכן הרשימות הוצגו ללא הסוגריים המרובעים.

כאשר דרוש לבצע חישובים על איברי הרשימה, פועלים על כל אחד מהם לפי הסדר. חישוב כזה נעשה אם-כן במחזורים, כשבכל מחזור מתבצעות פעולה, או פעולות, על איבר אחר ברשימה. הביצוע יכול להתחיל

באיבר הראשון, אחר-כך לעבור לאיבר השני לביצוע החישוב הנדרש, לעבור לאיבר שלאחריו וכך מאיבר לאיבר עד לאיבר האחרון.

ההתייחסות לאיבר הראשון ברשימה נעשית בעזרת פונקציית הבחירה FIRST. כך נוכל לבצע את החישובים באיבר הראשון שברשימה. בגמר הטיפול באיבר זה נוכל לפנות לרקורסיה, כאשר הקלט יהיה הפעם רשימה של שאר האיברים הדורשים טיפול. אלה הם כל איברי הרשימה פרט לראשון.

נגדיר הליך רקורסיבי **TALMID** לעיבוד של רשימה. ההליך מקבל רשימת תלמידים, ומדפיס את איבריה בזה אחר זה, כל איבר בשורה חדשה.

הגדרת TALMID

```
TO TALMID :RESHIMA
PR FIRST :RESHIMA
TALMID BF :RESHIMA
END
```

בקריאה להליך יש לרשום את השם TALMID עם רשימת התלמידים שתשמש קלט עבור :RESHIMA.

כאשר נרשום --->

```
?TALMID [DAN AVI GIL RONI]
```

נקבל את הפלט הבא:

```
DAN
AVI
GIL
RONI
```

כאשר תסתיים הדפסת כל איברי הרשימה נקבל את ההערה הבאה:

```
--------------------------------
FIRST DOESN"T LIKE [] AS INPUT
--------------------------------
```

במחזור הראשון מודפס האיבר הראשון 'DAN'. בפניה לרקורסיה הערך של :RESHIMA מבוצע 'BF :RESHIMA' כלומר - [AVI GIL RONI].

במחזור השני 'AVI' הוא האיבר הראשון, ולכן הוא מודפס. ושוב פונים למחזור חדש ונותנים 'BF :RESHIMA' של הרשימה הנוכחית. וכך ממחזור למחזור, כאשר בכל קריאה מופחת מהרשימה האיבר הראשון של אותו מחזור, עד שמסתיים הטיפול בכל אחד מאיברי הרשימה, ו-RESHIMA: הופכת לרשימה ריקה '[]'.

יש לדעת כי בלוגו כמו במתמטיקה, רשימה, גם אם אין בה אף לא איבר אחד, עדיין תקרא רשימה. מאחר ש-FIRST לא יכולה לבחור איבר מרשימה שאין בה איברים, מתערבת מערכת התוכנה של לוגו **ועוצרת את הביצוע**, ומציגה את ההודעה המתייחסת לרשימה ריקה. על מנת להמנע מהערות לוגו, נוכל לסיים את ביצוע ההליך בתנאי מסוים. תנאי מתאים להפסקת הביצוע במקרה זה הוא כאשר RESHIMA: הופכת לרשימה ריקה:

```
IF :RESHIMA = [] [STOP]
```

נוסיף אם-כן את משפט התנאי ונקבל הליך המסתיים ללא הערות.

```
TO TALMID :RESHIMA
IF :RESHIMA = [] [STOP]
PR FIRST :RESHIMA
TALMID BF :RESHIMA
END
```

חישוב של מספר איברים ברשימה

נדגים שני פתרונות שונים להגדרת פונקציה המחשבת את מספר האיברים אשר נמצאים ברשימה נתונה.

דוגמה אחת היא הפונקציה MISPAR1. בפתרון זה מתוסף בכל מחזור הערך '1' לערך הפונקציה של המחזור הנקרא. הערך הידוע שעליו נסתמך בחישוב הוא '0' (אפס) המציין את מספר האיברים ברשימה ריקה:

```
IF :LST = [] [OP 0]
```

נשתמש בערך זה כערך של המחזור האחרון. כלומר, כנקודת מוצא לחישוב כולו. במילים אחרות, ערך המחזור האחרון הוא '0' וערך מחזור n הוא '1 + (ערך מחזור n+1)'.

הגדרת MISPAR1

```
TO MISPAR1 :LST
IF :LST = [] [OP 0]
OP 1 + MISPAR1 BF :LST
END
```

בכל פניה רקורסיבית הקלט של LST: הוא 'BF :LST' של המחזור הקורא.

דוגמה אחרת היא הפונקציה MISPAR2 אשר נגדיר להלן. פונקציה זו מוגדרת עם משתנה מונה M:, אשר מקבל בתחילת החישוב ערך '0', ובכל מחזור מתוסף לו '1'. ערכו הסופי של המשתנה M:, המתקבל במחזור האחרון, מועבר למחזור שפנה אליו. המחזור שלפני האחרון מעביר ערך זה למחזור שקדם לו וכך עד להעברתו למחזור הראשון, אשר פולט אותו כערך של הפונקציה MISPAR2.

הגדרת MISPAR2

```
TO MISPAR2 :LST :M
IF :LST = [] [OP :M]
OP MISPAR2 (BF :LST) (:M + 1)
END
```

בכל פניה רקורסיבית ערך המשתנה M: גדל באחד. הקלט של הרשימה LST: של המחזור הנקרא הוא רשימת איברי LST: של המחזור הקורא **ללא האיבר הראשון** 'BF :LST'.

* * * * *

תרגילים

הערה: הפתרון עבור כמה מן התרגילים יכול להיות מורכב מכמה הליכים ו/או מכמה פונקציות.

1. הגדר הליך רקורסיבי המקבל רשימת שמות של תלמידים ומדפיס את המשפט הבא לכל אחד מן התלמידים שברשימה.

HELLO שם, IT IS NICE TO MEET YOU.

2. הגדר הליך רקורסיבי המקבל את רשימת חודשי השנה ומדפיס אותם בסדר הפוך, כלומר מן הסוף להתחלה.

3. הגדר הליך רקורסיבי המקבל שתי רשימות:
אחת היא רשימת ימי השבוע: [SUNDAY MONDAY ... SATURDAY]
השניה - רשימת המספרים הסידוריים: [1ST 2ND ... 7TH]
על ההליך להדפיס משפט כדוגמת המשפט הבא עבור לכל אחד מימי השבוע, בהתאמה:

THE 1ST DAY IN THE WEEK IS SUNDAY

4. הגדר הליך המקבל שם של עיר במשתנה CITY: ורשימת ערי בירה - במשתנה CAPITAL:. ההליך בודק אם הערך של CITY: נמצא ברשימה הנתונה של ערי הבירה, ומדפיס הודעה בהתאם.

הדרכה: השתמש בהוראה הבאה לפתרון הבעיה:

IF :CITY = FIRST :CAPITAL

5. הגדר הליך המקבל רשימת ארצות ומדפיס אותה פעמיים, בשתי עמודות מקבילות. בעמודה אחת יודפסו שמות הארצות לפי סדר הופעתם ברשימה, ובעמודה הימנית הם יודפסו בסדר הפוך, מן האחרון לראשון.
רמז: השתמש בשתי רשימות שיש בהן אותם ערכים.

6. הגדר פונקציה HIGHEST המקבלת רשימה של מספרים חיוביים, ומחזירה את הערך הגבוה ביותר המופיע ברשימה. לדוגמה, אם הקלט הוא '[19 23 120 1 57 44]', אז הפלט הוא '120'.

<u>הדרכה:</u> הגדר משתנה עזר H:, כך שההשוואה תיעשה בינו לבין כל אחד מאיברי הרשימה, ואשר בו ישמר הערך הגדול של כל מחזור. בתחילת הביצוע המשתנה H: יקבל ערך '0'.

7. השתמש באחת ההגדרות של MISPAR על מנת להגדיר פונקציה MEMUZA, לחישוב ממוצע של רשימת ציונים כלשהי.

8. הגדר פונקציה INDEX המקבלת רשימה של חברי קבוצה במשחק כדור-סל PLAYERS:, ופולטת את מספרו הסידורי של חבר מסוים HVR: כפי שהוא מופיע ברשימה הנתונה. אם אין חבר כזה, על הפונקציה להחזיר את הערך '0'.

9. הגדר פונקציה HAVER המקבלת רשימת חברים LST: ומספר N:, ומחזירה את האיבר הנמצא במקום ה-N: ברשימה.

* * * * *

פונקציות בדיקה

מילה מורכבת מתווים, או שהיא ריקה. למילה אשר איננה ריקה יש תו ראשון ואחריו המשך המילה. ואפשר לציין את התו האחרון שלה ואת חלק המילה של פניו.

כאשר נתונה המילה 'APPLE' כערך של המשתנה WRD:, נקבל:

האות 'A'	<---	FIRST :WRD
האות 'E'	<---	LAST :WRD
חלק המילה 'PPLE'	<---	BF :WRD
חלק המילה 'APPL'	<---	BL :WRD

נוכל להגדיר הליך המטפל בתווי המילה בזה אחר זה באופן

רקורסיבי. כל מחזור מתייחס לתו הראשון של המילה שנמצאת במשתנה WRD: למשל, על-יד הביטוי 'FIRST :WRD'. ערך המשתנה WRD: של המחזור הבא יהיה 'BF :WRD' של המחזור הקורא, עד אשר נקבל ב-WRD: **מילה ריקה** '"' (Null). נוכל לבדוק אם לפנינו מילה ריקה ולסיים את התהליך.

```
IF :WRD = " [ ... STOP]
```

כשם שפונקציות הבחירה אינן מקבלות **רשימה ריקה** כקלט, כן אין הן מקבלות **מילה ריקה** כקלט. במקרה זה למשל, נקבל עבור הפונקציה BF, את ההערה הבאה:

```
--------------------------------
BF DOESN'T LIKE   AS INPUT
--------------------------------
```

כדי להמנע מהודעת השגיאה כדאי להשתמש **בפונקציות בדיקה**, ולהמשיך את התכנית בהתאם לתוצאה.

הפרדיקט EMPTYP

פרדיקט זה מקבל קלט כלשהו, כמו רשימה, מילה או מספר (המורכב גם הוא מרצף של תווים), ובודק אם קלט זה הוא "ריק" או שאינו ריק.

לכן נוכל לרשום `IF EMPTYP :WRD`

במקום `IF :WRD = "`

ובמקום `IF :RESHIMA = []`

נוכל לרשום `IF EMPTYP :RESHIMA`

הפרדיקט LISTP

הפרדיקט מקבל קלט ובודק אם קלט זה מהווה רשימה.

לדוגמה: `IF LISTP :WRD`

הפרדיקט WORDP

הפרדיקט מקבל קלט ובודק אם קלט זה מהווה מילה.

לדוגמה: IF WORDP :WRD

* * * * *

תרגילים (המשך)

10. הגדר פונקציה LENGTH המחשבת את אורכה של מילה כלשהי. כלומר, יש למצוא את מספר התווים המרכיבים את המילה.

11. הגדר פונקציה LONGER המקבלת שתי מילים, בודקת אם WRD1: ארוכה יותר מ-WRD2:, ומחזירה הודעה מתאימה.
<u>הנחיה:</u> הפונקציה LONGER יכולה להשתמש בפונקציה LENGTH, שהוגדרה בתרגיל קודם.

12. הגדר הליך המקבל רשימה של מספרים בשבר עשרוני ומציג אותם בטור, זה תחת זה, עם **יישור הנקודה העשרונית**.

<u>הדרכה:</u> קבע תחילה עמודה, למשל העמודה 11, שבה תופיע הנקודה העשרונית. אם מספר הספרות של החלק השלם הוא 6, יש להדפיס את המספר לאחר הדפסת 4 רווחים. לכן יש לחשב תחילה כמה רווחים דרוש להדפיס משמאל.

13. הגדר הליך המקבל מספר המורכב מתשע (9) ספרות לכל היותר, מדפיס אותו ואת **סכום** ספרותיו. אם אורך הקלט הוא יותר מ-9 תווים, יש להפסיק את החישוב ולתת הודעה כי מספר הספרות לא מתאים.

14. הגדר פונקציה POSI המקבלת מילה במשתנה MILA: ואות אחת במשתנה OT:, ומחזירה את מקום ההופעה הראשון של האות במילה. אם אות זו איננה חלק מן המילה הנתונה, על הפונקציה להחזיר את הערך '0'.

15. FIRST הינה פונקציה המחזירה את האיבר הראשון של רשימה כלשהי. נסה להגדיר פונקציה SECOND שהפלט שלה יהיה האיבר השני של רשימה כלשהי.

16. הגדר פונקציה BLAST המחזירה את האיבר שלפני האחרון ברשימה כלשהי.

17. בהנחה שלא קיים הפרדיקט EMPTYP, הגדר אותו בעצמך.

18. הגדר פרדיקט SINGLEP הבודק אם קיים רק איבר אחד ברשימה, או אם קיים רק תו אחד במילה.

* * * * *

הפונקציה WORD

הפונקציה WORD מקבלת שתי מילים ומחברת אותן למילה אחת.

המבנה הכללי: **WORD מילה1 מילה2**

אם הערך של X: הוא המילה 'BA', והערך של Y: הוא המילה 'NANA', אז ערך הביטוי 'WORD :X :Y' הוא 'BANANA'. הערך של X: מתחבר עם הערך של Y: ויוצר מילה אחת בהתאם לסדר שבו הם רשומים. על-כן, ערך הביטוי 'WORD :Y :X' הוא 'NANABA'.

הפונקציה WORD יכולה לפעול על יותר מאשר שתי מילים, אך לשם כך יש צורך לתחום את כל הביטוי בסוגריים עגולים.

המבנה הוא: (WORD מילה1 מילה2 ... מילהn)

נתון הביטוי הבא: (WORD :X :Y 1 :X :Y 2 "BA "NANA 3)

הערך שלו הוא: BANANA1BANANA2BANANA3

נגדיר פונקציה RVRS המקבלת מילה, ומחזירה אותה בסדר הפוך (כלומר, היפוך סדר התווים המרכיבים אותה).

הגדרת RVRS

```
TO RVRS :MILA
IF EMPTYP :MILA [OP " ]
OP WORD (LAST :MILA) (RVRS BL :MILA)
END
```

כאשר נרשום ---> `?PR RVRS "ABCD`

הפלט יהיה: `DCBA`

החישוב נעשה באופן רקורסיבי. כל מחזור מציב את האות האחרונה של הקלט שלו בתחילת הערך המתקבל מתוצאת החישוב של המחזור הנקרא על-ידו:

```
OP WORD (LAST :MILA) (RVRS BL :MILA)
```

	התוצאה	הפעולה
מחזור א'	DCBA	WORD "D RVRS "ABC
מחזור ב'	CBA	WORD "C RVRS "AB
מחזור ג'	BA	WORD "B RVRS "A
מחזור ד'	A	WORD "A RVRS "

הסימן "" בלבד מציין מילה ריקה. הערך הידוע, אשר משמש כנקודת מוצא לחישוב התרגיל מתקבל **במחזור האחרון**, כאשר יש מילה ריקה:

```
IF EMPTYP :MILA [OP " ]
```

ניתן להגדיר משתנה REV:, שערכו בתחילת הביצוע יהיה המילה הריקה "". בקריאה לרקורסיה נחבר לסוף המילה REV: את האות האחרונה של הנתון MILA: השייך לאותו מחזור, ואילו הערך של MILA: יהיה

'BL :MILA'. הביצוע מסתיים כאשר MILA: הופכת לריקה, ואז מוחזר הערך של REV:.

הגדרת REVERSE

```
TO REVERSE :MILA :REV
IF EMPTYP :MILA [OP :REV]
OP REVERSE (BL :MILA) (WORD :REV LAST :MILA)
END
```

כפי שרואים, המשתנה REV: של כל מחזור חדש מקבל את ערכו של הביטוי הבא:

```
WORD :REV LAST :MILA
```

כך למעשה משתנה ערכו של REV: ממחזור למחזור, עד שלא נשאר כל תו שהוא ב-MILA: ואז כאמור, מחזירה REVERSE את הערך של REV:.

בקריאה הראשונה, הערך של MILA: הוא 'ABCD' ו-REV: שווה NULL.
בקריאה השניה - MILA: תהיה 'ABC' ו-REV: = 'D'.
בקריאה השלישית - MILA: = 'AB' ו-REV: = 'DC'.
ברביעית - MILA: = 'A' ו-REV: = 'DCB'.
בקריאה האחרונה - MILA: = (NULL) ו-REV: = 'DCBA'.

לסיום - מקבלים את המילה 'ABCD' כשהיא הפוכה: 'DCBA'.

הפרדיקט BEFOREP
(קיים בגירסת APPLE LOGO II)

פרדיקט זה מקבל שתי מילים, ומחזיר ערך אמת אם המילה הראשונה נמצאת לפני המילה השניה, על פי הסדר המילוני.

לדוגמה, אם נרשום ---> PR BEFOREP "DONALD "MICKEY
נקבל: TRUE

* * * * *

תרגילים (המשך)

19. הגדר פונקציה PALINDROM המקבלת מילה ובודקת אם התווים מופיעים בה באופן סימטרי, כלומר שקריאתה משמאל לימין תהיה זהה עם קריאתה מימין לשמאל. למשל המילים 'NOON' או 'LEVEL' ממלאות תנאי זה.

<u>הדרכה:</u> בקריאה לרקורסיה ערך המילה יהיה:

```
BF (BL :MILA)
```

20. הגדר הליך המקבל את המילה 'ABRACADABRA' ומדפיס אותה כשבכל שורה מופחת תו אחד, עד לתו האחרון שנשאר.

לדוגמה:

```
ABRACADABRA
BRACADABRA
...
RA
A
```

21. נסה להגדיר הליך המקבל את המילה 'ABRACADABRA' (כמו בשאלה הקודמת), ומדפיס בשורה הראשונה את האות האחרונה 'A', בשורה השניה את 2 האותיות האחרונות 'RA', בשורה השלישית את 3 האותיות האחרונות 'BRA', וכן הלאה עד שבשורה האחרונה מודפסת כל המילה בשלימותה.

* * * * *

תווים בקוד ASCII

תו יכול להיות אחת האותיות (A B...Z), אחת הספרות (0 1...9), אחד הסימנים של 4 פעולות החשבון, פעולות יחס, סימני פסוק, או כל סימן מוגדר אחר (@ $...) וגם תו-רווח ' '.

המחשב מזהה כל תו על פי קוד מסוים, שהוא הערך המספרי שלו. השימוש המקובל במחשבים אישיים הוא על פי קוד ASCII, אשר פותח בארה"ב, והוא מציין בראשי תיבות את הביטוי:
American Standard Code for Information Interchange - ASCII

הפונקציה ASCII

פונקציה זו, הנקראת על שם הקוד עצמו, מקבלת ערך שהוא **מילה**, ומחזירה את ערך הקוד של התו הראשון המופיע במילה.

אם נרשום ---> ?PR ASCII "A
או נכתוב ---> ?PR ASCII "AVI
נקבל בשני המקרים את הקוד של האות 'A': 65

מספר זה הוא הערך של האות 'A' בטבלת קוד ASCII. 'A' הינה האות הראשונה באותיות האלף-בית האנגלי, ומכיון שאותיות אלו קיבלו את ערכי הקוד על פי הסדר שלהם, נמצא שערך הקוד של האות 'B' הוא 66, של האות 'C' הוא 67 וכן הלאה.

הפונקציה CHAR

הפונקציה CHAR הפוכה לפונקציה ASCII. היא מקבלת ערך מספרי הנע בין 0 ל-255, ומחזירה את התו המיוצג על-ידי אותו מספר.

כאשר נרשום ---> ?PR CHAR 65
תודפס האות 'A' שהיא בעלת אותו קוד: A

הערות בתכנית

בשפות תכנות שונות קיימת דרך לציין הערות בגוף התכנית. המחשב אינו מתייחס אליהן, ואינו מקבל אותן כהוראות שעליו לבצע. הן מיודות לנוחות המתכנת בלבד.

בלוגו לא מוגדרת מילה המאפשרת כתיבת **הערה** (Remark). נסה להגדיר בעצמך **הליך REM** אשר מקבל **רשימה** כלשהי. כאשר מתבצעת קריאה להליך, המחשב אינו מתייחס לתוכן הרשימה, ולכן אינו מבצע כל פעולה שהיא על הרשימה הזו.

לדוגמה, נכתוב הערה בתכנית ---> REM [This is my Program]

* * * * *

תרגילים (המשך)

22. הגדר פונקציה CONCAT המקבלת **רשימת** מילים, (אשר יכולות להכיל תו אחד או יותר), ומחזירה **מילה** המורכבת מהאיברים שברשימה.
    תוכל לכלול בהגדרה הערה באמצעות ההליך REM שהגדרת, כדי לציין את תפקיד הפונקציה, ואת שלבי הפתרון.

23. כתוב הליך המקבל **רשימת** שמות, ומדפיס את האות הראשונה של כל שם (כלומר, של כל איבר).

<u>הדרכה:</u> PR FIRST (FIRST :RESHIMA)

24. כתוב תכנית המקבלת רשימת שמות, ומדפיסה רק את אלה, אשר האות הראשונה שלהם היא מ-'L' ואילך באלף-בית הלטיני.

25. לקורס שחיה מתקבלים ילדים שגילם אינו נמוך משמונה שנים.
השיעורים מתקיימים בשלושה ימים בשבוע:

יום א' עבור ילדים שהאות הראשונה בשם שלהם היא מ-A עד H,
יום ג' עבור ילדים שהאות הראשונה בשם שלהם היא מ-I עד P,
יום ה' עבור ילדים שהאות הראשונה בשם שלהם היא מ-Q עד Z.

כתוב תכנית המקבלת רשימה ובה שמות הילדים וגיליהם לפי הסדר הבא:

[שם1 גיל1 שם2 גיל2 ... ... שםח גילח]

התכנית תציג על המסך אם הילד התקבל או לא, ואם כן - באיזה יום הוא ילמד.

<u>הדרכה:</u> ניתן להתייחס לאיבר השני ברשימה על-ידי הביטוי הבא:

FIRST (BF :RESHIMA)

האיברים הנותרים לטיפול יהיו:

BF (BF :RESHIMA)

26. הגדר פרדיקט LIFNEIP המקבל שני שמות :NAME1 ו-:NAME2 ובודק אם :NAME1 נמצא לפני :NAME2 על פי הסדר המילוני. אם גירסת לוגו שברשותך כוללת את הפונקציה BEFOREP, אל תשתמש בה.

27. כתוב פונקציה המקבלת רשימת שמות, מוצאת את השם הגדול ביותר על פי הסדר המילוני ומחזירה אותו.

פרק 12

עיבוד רשימות

בשיעור הקודם למדנו על פונקציות הבחירה המשמשות לטיפול ברשימות ובמילים. הסברנו כיצד הן פועלות, מהו הקלט הדרוש להן ומהו הפלט שניתן להפיק מהן. הכרנו גם את הפונקציה WORD המרכיבה מילה על-ידי חיבור של מספר מילים למילה אחת.

פונקציות הרכבה

ההוראה SE (SENTENCE), הינה אחת הפונקציות המשמשות להרכבת רשימות. היא **מאחדת** דברים לרשימה אחת, ולכן אפשר באמצעותה **לאחד** לרשימה אחת נתונים קבועים, תוכן של משתנים ותוכן של רשימות.

נלמד עתה פונקציות נוספות המרכיבות רשימות.

הפונקציה LIST

המבנה הכללי: **LIST דבר**

או: **LIST 1דבר 2דבר**

הפונקציה LIST מקבלת דבר אחד או שני דברים ושמה אותם באריזה אחת, דהיינו, ברשימה אחת.

לדוגמה, אם למשתנה X: יש הערך '57' ולמשתנה A: יש הערך 'ABC'. ערך הביטוי 'LIST "XYZ :A' הוא '[XYZ ABC]'. במקרה זה הפונקציה LIST בנתה רשימה המכילה את הנתון 'XYZ' והערך של A:.

ערך הביטוי יהיה שווה, אם במקום LIST נרשום: 'SE "XYZ :A'

אבל, כאשר אחד הדברים הוא רשימה, כמו בביטוי הבא:

LIST [40 + 17 =] :X

ערכו הוא: [[40 + 17 =] 57]

רשימת הקלט המכילה תרגיל חשבוני, מופיעה ברשימת הפלט כפי שהיא, כדבר נפרד. נמצא, שמספר הדברים ש-LIST מקבלת, הוא מספר **האיברים** שיהיו בתוך הרשימה שהיא בונה.

לדוגמה, הפונקציה LIST קיבלה 2 דברים:

- דבר1 = [40 + 17 =]
- דבר2 = X:

ברשימת הפלט נקבל 2 דברים:

- איבר1 = [40 + 17 =]
- איבר2 = הערך של X:

לא כך, אם נשתמש ב-SE:

SE [40 + 17 =] :X

במקרה זה, ערך הביטוי יהיה: [40 + 17 = 57]

כאן מאחדת SE לרשימה אחת את האיברים המופיעים ברשימת התרגיל החשבוני יחד עם הערך של X:. מספר האיברים המתקבלים ברשימה החדשה הוא כמספר איברי הרשימה המופיעה כקלט, ועוד נתון אחד X:.

הפונקציה SE קיבלה 2 דברים:

- דבר1 = [40 + 17 =]
- דבר2 = X:

מספר האיברים, שהפונקציה SE החזירה בתוך הרשימה, שונה מאשר במקרה הקודם:

- איבר1 = '40'
- איבר2 = '+'
- איבר3 = '17'
- איבר4 = '='
- איבר5 = '57'

כזכור, **דבר** יכול להיות תו, מילה או רשימה. לכן, גם **איבר** של רשימה יכול להיות תו, מילה או רשימה.

LIST יכולה לקבל יותר משני דברים ולבנות אותם ברשימה אחת. לשם כך יש לתחום את כל המשפט בסוגריים עגולים:

```
------------------------------
(LIST  דבר1  דבר2 ... דברת)
------------------------------
```

לדוגמה, ערך הביטוי (LIST :A [DEF GHI 8] [KLM NN] :X "XYZ)

הוא: [ABC [DEF GHI 8] [KLM NN] 57 XYZ]

שים לב להבדל בין פלט זה לבין הפלט המתקבל באמצעות SE.

הערך של הביטוי, (SE :A [DEF GHI 8] [KLM NN] :X "XYZ)

הוא: [ABC DEF GHI 8 KLM NN 57 XYZ]

הפונקציה FPUT {FirstPUT}

הפונקציה FPUT מקבלת שני קלטים, שהראשון בהם הוא דבר כלשהו והשני הוא רשימה, ומציבה את הדבר **בתחילת** הרשימה.

כאשר נרשום ---> ?SHOW FPUT 17 [JULY 1987]

נקבל: [17 JULY 1987]

דוגמה נוספת <--- ?SHOW FPUT [A BB CCC] [[1 22] [333] 444]

נקבל את הרשימה הבאה: [[A BB CCC] [1 22] [333] 444]

יש לשים לב לכך שהאיברים ברשימת הקלט הם:

'[1 22]' ,'[333]' ,'444'

ברשימת הפלט **התוסף** להם האיבר '[A BB CCC]' והוצב בראש.

אם היינו משתמשים ב-SE במקום ב-FPUT, היינו מקבלים פלט זהה בדוגמה הראשונה, אך בדוגמה השניה היינו מקבלים את הרשימה הבאה:

[A BB CCC [1 22] [333] 444]

הפונקציה SE איחדה את איברי הרשימה '[A BB CCC]' יחד עם איברי רשימת הקלט השניה לרשימה אחת.

הפונקציה LPUT {LastPUT}

הפונקציה LPUT מקבלת שני קלטים, שהראשון הוא דבר והשני - רשימה, ומציבה את הדבר **בסוף** הרשימה.

דוגמה <--- ?SHOW LPUT [] [[1 2] [1] [2]]

הפלט יהיה: [[1 2] [1] [2] []]

הרשימה הריקה הופיעה כאיבר אחרון ברשימת הפלט.

נגדיר פונקציה המקבלת שני קלטים: רשימת שמות של בנים BOYS:, ורשימה אחרת GIRLS: ובה שמות של בנות. על הפונקציה לבנות רשימות של זוגות, כך שהאיבר הראשון בכל רשימה כזו יהיה שם בן, והאיבר השני יהיה שם של בת. על הפונקציה להחזיר רשימה אחת שמכילה את כל הרשימות שהיא בנתה.

הגדרת COUPLE

```
TO COUPLE :BOYS :GIRLS
IF OR EMPTYP :BOYS EMPTYP :GIRLS [OP []]
OP FPUT (LIST FIRST :BOYS FIRST :GIRLS)
                 (COUPLE BF :BOYS BF :GIRLS)
END
```

כאשר נרשום ---> ?PR COUPLE [DAN NIR ZIV] [ORA TAL RUT]

נקבל את הפלט הבא: [DAN ORA] [NIR TAL] [ZIV RUT]

<u>הסבר</u>:

בתרגיל זה יש לבנות רשימה לכל צמד, ולכן השתמשנו בביטוי -

(LIST FIRST :BOYS FIRST :GIRLS)

במחזור הראשון, ערך הביטוי הוא [DAN ORA]. ערך זה מוצב באמצעות הפונקציה **FPUT** בראש **רשימת הפלט** של המחזור הבא, כאילו רשמנו:

OP **FPUT** [DAN ORA] **(COUPLE BF :BOYS BF :GIRLS)**

עתה עובר הביצוע למחזור שני, כשהקלט הוא:

[NIR ZIV] עבור :BOYS

[TAL RUT] עבור :GIRLS

הפונקציה בונה רשימה לצמד [NIR TAL] ומציבה אותה בתחילת **רשימת הפלט** של המחזור השלישי. וזה, כאילו רשמנו:

OP **FPUT** [NIR TAL] **(COUPLE BF :BOYS BF :GIRLS)**

במחזור שלאחר מכן:-

הערך '[ZIV]' עובר ל-BOYS:

הערך '[RUT]' עובר ל- GIRLS:

במחזור זה מוצבת רשימת הצמד [ZIV RUT] בתחילת **רשימת הפלט** של

המחזור האחרון. ערך הפלט של המחזור האחרון הוא הרשימה הריקה '[]', אשר משמשת כנקודת מוצא לחישוב כולו. ערך זה מוחזר כאשר אחת משתי הרשימות הופכת לריקה:

```
IF OR EMPTYP :BOYS EMPTYP :GIRLS [OP []]
```

ואז כאילו רשמנו -

```
OP FPUT [ZIV RUT] []
```

תוצאת הביטוי '[[ZIV RUT]]', היא רשימת הפלט של המחזור השלישי.

נציב ערך זה במקום המתאים בביטוי, שעל המחזור השני לחשב:

```
OP FPUT [NIR TAL] [[ZIV RUT]]
```

התוצאה תהיה:

```
[[NIR TAL] [ZIV RUT]]
```

שוב נציב את ערך התוצאה לביטוי שיש לחשב על-ידי המחזור הראשון:

```
OP FPUT [DAN ORA] [[NIR TAL] [ZIV RUT]]
```

והפלט יהיה:

```
[[DAN ORA] [NIR TAL] [ZIV RUT]]
```

רשימה זו היא התוצאה הסופית, או ערך הפונקציה.

* * * * *

תרגילים

1. הגדר פונקציה המקבלת רשימת צבעים, ומחזירה רשימה שבה האיבר הראשון של הקלט יהיה האיבר האחרון של הפלט.

אם הקלט הוא - [YELLOW BLUE GREEN RED]

הפלט יהיה - [BLUE GREEN RED YELLOW]

2. הגדר פונקציה HAFUCH המקבלת רשימת נתונים, ומחזירה רשימה שבה נתונים אלה מסודרים בסדר הפוך. כלומר, האיבר הראשון יהיה האחרון, האיבר השני יהיה האיבר שלפני האחרון ... והאחרון יהיה הראשון.

3. הגדר פונקציה PIRUK המקבלת מילה MILA: כקלט, ומחזירה רשימה שאיבריה הם התווים המרכיבים את המילה.

לדוגמה, אם הקלט הוא: 'OTIOT', הפלט יהיה: '[O T I O T]'.

4. במבחן על-פה הספיק המורה לבחון רק 5 תלמידים. הגדר פונקציה המקבלת את רשימת כל תלמידי הכתה ומחזירה כפלט את רשימת התלמידים שעדיין לא נבחנו. רשימה זו תכיל את שמות התלמידים המופיעים מהמקום השישי ואילך.

5. הגדר פונקציה DEL המקבלת LIST:, שהיא רשימת מועמדים לתפקיד כלשהו, ושם של מועמד NAME: שנבחר לתפקיד. הפונקציה מחזירה את רשימת המועמדים שלא נבחרו.

6. הגדר פונקציה CHANGE המקבלת רשימה של דיירי בניין DRM: ומחזירה רשימה חדשה, בה מוחלף שמו של דייר שעזב OLDN: בשמו של הדייר החדש NEWN:.

7. לכיתה מסוימת הצטרף תלמיד חדש NEWSTUD:. הגדר פונקציה INSERT, שמקבלת רשימת התלמידים STUDENTS: כשהיא מסודרת על פי האלף-בית, ופולטת רשימה חדשה, כאשר שמו של התלמיד החדש מופיע במקום המתאים.

8. מסופר על 21 מורדים בימי המרד הגדול ברומאים, אשר החליטו להתאבד זה אחר זה. הם ישבו במעגל במקומות ממוספרים, וקבעו ביניהם להתחיל בספירת הנוכחים וכל אדם שביעי יתאבד בתורו. הספירה הראשונה התחילה מזה שנקבע כיושב במקום הראשון. הספירה הבאה התחילה מזה היושב אחרי המתאבד האחרון (הספירה מתייחסת רק לנותרים בחיים).

כתוב פונקציה אשר תמצא את מקומו של זה שניצל ממוות. הצג זאת כפלט עם הודעת הסבר.

* * * * *

קבוצות

קבוצה (Set) מכילה רשימה של איברים. לקבוצה תכונות שונות, אשר נחזור עליהן בקצרה:

א. אין בקבוצה חזרות, כל איבר מופיע בה פעם אחת בלבד (אך לא כך ברשימה, שבה איבר יכול להופיע פעמים אחדות).

נתונות שתי רשימות:

L1 - [11 12 13 12 15]

ו-L2 - [11 12 13 14 15]

נבדוק אם אלו הן קבוצות:

- L1 **אינה** קבוצה, האיבר **שערכו** 12 נמצא בה פעמיים.
- L2 **כן** מהווה קבוצה.

ב. אין חשיבות לסדר שבו רשומים איברי הקבוצה.

לדוגמה, את קבוצת המספרים הזוגיים מ-1 עד 10, ניתן לייצג

על-ידי: [2 4 6 8 10]

גם על-ידי: [10 8 6 4 2]

וגם על-ידי כל רשימה אחרת, המכילה את אותם הערכים.

ג. קבוצה שאין בה אף לא איבר אחד היא **קבוצה ריקה** (Empty Set).

בדיקת שייכות של איבר לרשימה

בתרגיל 4 שבפרק הקודם צריך היה להגדיר הליך, הבודק אם העיר :CITY מופיעה כאחד האיברים של הרשימה :CAPITAL. נלמד עתה את הפונקציה שניתן באמצעותה לפתור תרגילים מן הסוג הזה.

הפרדיקט MEMBERP

הפרדיקט MEMBERP מקבל שני קלטים: **איבר ורשימה**. הפלט הוא 'אמת' אם איבר זה **שייך** לרשימת הקלט, ואם איננו שייך - הפלט הוא 'שקר'.

המבנה הכללי: **MEMBERP דבר רשימה**

לדוגמה, אם נרשום ---> יוצג על המסך:

```
?PR MEMBERP "AVI [GIL URI DOV ELY]
FALSE
```

עוד דוגמה ---> התשובה היא:

```
?PR MEMBERP [3 [87]] [2 4 [3 [87]] [5 90]]
TRUE
```

שים לב: בדוגמה השניה האיברים של רשימת הקלט LIST הם:

א. '2',
ב. '4',
ג. '[3 [87]]',
ד. '[5 90]'.

על-כן, האיבר ("דבר") השווה ל-'[3 [87]]' **כן** שייך לרשימה.

בדיקת יחס בין קבוצות

לבד מבדיקת השייכות של איבר מסוים לקבוצה נתונה, קיימות במתמטיקה פעולות לביצוע **בדיקות יחס** בין קבוצות. כאשר נתונות שתי קבוצות A ו-B, נוכל לומר אם הן שוות, אם אחת חלקית לשניה, או שהן קבוצות זרות. קיימות פעולות אלגבריות לביצוע **חישובים** על קבוצות, כמו למשל איחוד, חיתוך והפרש בין קבוצות. פעולות אלו אינן מוגדרות בלוגו, אך יש באפשרותנו להגדירן.

נביא לדוגמה הגדרה של הפונקציה UNION, **לאיחוד** שתי קבוצות A ו-B (A U B). UNION מקבלת עבור שני המשתנים A: ו-B: שתי רשימות שהן קבוצות, ומחזירה את **קבוצת** כל האיברים הנמצאים ב-A, ב-B או בשתיהן גם יחד.

הגדרת UNION

```
TO UNION :A :B
IF EMPTYP :A [OP :B]
TEST MEMBERP (FIRST :A) :B
IFT [OP UNION BF :A :B]
IFF [OP FPUT (FIRST :A) (UNION BF :A :B)]
END
```

כאשר נרשום ---> ?PR UNION [23 7 17] [33 23 6 7]

נקבל את התוצאה הבאה: 17 33 23 6 7

<u>הסבר</u>:

כנקודת מוצא, הפלט של המחזור האחרון יכלול את כל האיברים הנמצאים בקבוצה B:, וזאת כאשר לא נמצא אף איבר בקבוצה A: -

```
IF EMPTYP :A [OP :B]
```

על פי הדוגמה הזו, הפלט של המחזור האחרון הוא: [33 23 6 7].

הפונקציה בודקת בכל מחזור, אם האיבר הראשון של הקבוצה A: שייך לקבוצה B: -

```
TEST MEMBERP (FIRST :A) :B
```

- אם כן, אין מציבים איבר זה ברשימת הפלט, כדי למנוע הופעה חוזרת של אותו איבר. הביצוע עובר למחזור חדש לבדיקת האיבר הבא בתור בקבוצה A: -

```
OP UNION BF :A :B
```

- אם האיבר איננו שייך לקבוצה B:, מציבה UNION איבר זה בראש תוצאת הפלט המתקבלת מהמחזור שנקרא על-ידי המחזור הנוכחי.

```
OP FPUT (FIRST :A) (UNION BF :A :B)
```

הביצוע מסתיים כאשר הרשימה A: הופכת לרשימה ריקה.

* * * * *

תרגילים (המשך)

9. הגדר פונקציה SET המקבלת רשימת איברים, אשר יכולות להיות בה חזרות. הפונקציה מחזירה את רשימת כל האיברים **כקבוצה**, דהיינו ללא חזרות.

<u>הדרכה:</u> אם האיבר הראשון לא מופיע ברשימת שאר האיברים, יש להציב איבר זה ברשימת הפלט.

10. הגדר פרדיקט SETP הבודק אם הרשימה LST: מהווה קבוצה.

11. הגדר פרדיקט EQSETP הבודק אם שתי הקבוצות L1: ו-L2: שוות.

<u>הערה:</u> שתי קבוצות הן שוות, כאשר כל אחת מהן היא חלקית לשניה. זכור, כי שתי קבוצות הן זהות גם אם איבריהן אינם מופיעים באותו סדר.

12. שתי כיתות לומדות מקצוע מסוים, כל אחת עם מורה אחר. הכיתה הראשונה למדה בסמסטר הראשון קבוצה של n נושאים, והכיתה השניה הספיקה ללמוד באותו פרק זמן קבוצה של m נושאים (n יכול להיות שונה מ-m).

אפשר שלא כל הנושאים בקבוצה האחת יהיו זהים (חופפים) לאלה שבקבוצה השניה.

א. כתוב פונקציה המקבלת שתי קבוצות: קבוצת הנושאים של הכיתה הראשונה LSTA:, וקבוצת הנושאים של הכיתה השניה LSTB:. היא מחזירה רשימה, שבה קבוצת הנושאים הנמצאים רק ב-LSTA: אך אינם ב-LSTB:.

לדוגמה, LSTA: ---> [NOSE1 NOSE3 NOSE4 NOSE5 NOSE7]

ו-LSTB: ---> [NOSE1 NOSE2 NOSE5 NOSE6]

הפלט יהיה --> [NOSE3 NOSE4 NOSE7]

ב. כתוב פונקציה המקבלת שתי קבוצות כמו בתרגיל 12א' ומחזירה רשימה של קבוצת הנושאים החופפים. כלומר, הנושאים הנמצאים **גם** ב-LSTA: **וגם** ב-LSTB:, ולא רק באחת מהן.

על פי הדוגמה הקודמת, הפלט יהיה: [NOSE1 NOSE5]

ג. כתוב פונקציה המקבלת שתי קבוצות כמו בתרגיל 12א', ומחזירה רשימה של קבוצת הנושאים הנמצאים בכל אחת מהקבוצות LSTA: ו-LSTB: אך אינם חופפים. כלומר, נקבל רשימה של כל הנושאים הרשומים רק ב-LSTA: יחד עם הנושאים הרשומים רק ב-LSTB:.

לפי הדוגמה נקבל: [NOSE2 NOSE3 NOSE4 NOSE6 NOSE7]

* * * * *

יצירת רשימות

הפונקציה SIDRAFIB משמשת ליצירת רשימה של NUM: האיברים הראשוניים מסידרת פיבונצ'י (Fibonacci). פונקציה זו משתמשת בפונקציה FIB, שהוגדרה באחד הפרקים הקודמים, על מנת שזו האחרונה תחשב לה את האיבר ה-N: בסידרה.

הגדרת SIDRAFIB

```
TO SIDRAFIB :NUM
IF :NUM < 1 [OP []]
OP LPUT (FIB :NUM) (SIDRAFIB :NUM - 1)
END
```

הפונקציה SIDRAFIB מקבלת קלט שהוא ערך בודד, ומחזירה פלט שהוא רשימה.

כאשר נרשום --- ?SHOW SIDRAFIB 7

נקבל רשימה ובה 7 איברים ראשונים של הסידרה: [1 1 2 3 5 8 13]

בכל מחזור, הפונקציה SIDRAFIB משתמשת בהוראה LPUT

כדי להציב בסוף ההוראות של **רשימת הפלט** של המחזור הבא ('SIDRAFIB :NUM - 1'),

את האיבר ה-N: של הסידרה ('FIB :NUM'), כערך התחלתי לחישוב.

הפונקציה מחזירה את הרשימה הריקה '[]' עבור מספר איברים הקטן מ-1: IF :NUM < 1 [OP []]

* * * * *

תרגילים (המשך)

13. הגדר פונקציה לבניית רשימה בת N: איברים של סידרת מספרים, שבה ההפרשים מהווים סידרה של המספרים האי-הזוגיים הטבעיים. כלומר, אם **האיבר הראשון** בסידרה המקורית הוא הנתון X:, **האיבר השני** הוא X+1:, **האיבר השלישי** שווה **לאיבר השני** + 3, **האיבר הרביעי** שווה **לאיבר השלישי** + 5, וכן הלאה.

14. נתונה הפונקציה $Y = X^2 - X - 1$
הגדר פונקציה המחזירה רשימה של זוגות הערכים השלמים X ו-Y, עבור ערכי X בין (‎-3) לבין (4), בהפרשים של 1. סה"כ תקבל 8 איברים המופיעים כרשימות.

15. לפניך 7 האיברים הראשונים של סידרת המספרים הטבעיים SDR:.
1, 3, 7, 12, 18, 26, 35, ...
ההפרשים של SDR: מהווים אף הם סידרה של מספרים:
2, 4, 5, 6, 8, 9, ...
כתוצאה, **איחוד** של שתי סדרות אלה מהווה את **קבוצת** המספרים הטבעיים (קבוצה, כלומר שכל איבר בה מופיע פעם אחת בלבד).

הגדר פונקציה לחישוב 100 האיברים הראשונים של SDR:.

16. הגדר פונקציה לבניית רשימה של MIS: מספרים אקראיים, בתחום של (1...99).

17. הגדר פונקציה הבונה מילה בת N: **אותיות** הנבחרות באופן אקראי.

* * * * *

רשימה ממויינת

כאשר נתונה רשימה של נתונים בעלי תכונה משותפת, כמו רשימת רחובותיה של עיר, רשימת פרחי-בר, או רשימת תאריכים היסטוריים, נמצא שנתונים אלה מופיעים לפי סדר מסוים. על פי סדר זה ניתן לקבוע איזה נתון מבין שני נתונים בא קודם. רשימה כזו נקראת **רשימה ממויינת**.

בדרך כלל בונים את הרשימה תוך כדי קליטת נתונים ללא סדר כלשהו ואחר-כך ממיינים אותה לפי סדר רצוי. מבחינים במיון על פי סדר עולה, שבו כל איבר קטן יותר מזה הבא אחריו, ומיון בסדר יורד שבו האיבר הגדול נמצא לפני האיבר הקטן ממנו.

נגדיר להלן תכנית למיון רשימת נתונים מספריים לפי סדר עולה. התכנית מורכבת משתי פונקציות: SORT ו-SORTING. הפונקציה SORT גורמת לכך שהאיבר בעל הערך הגדול ביותר יימצא בסוף הרשימה. הפונקציה SORTING מציבה בתהליך רקורסיבי את האיבר המקסימלי הזה, שחושב על-ידי SORT, בסוף רשימת הפלט של שאר האיברים של המחזורים הבאים.

הגדרת SORT

```
TO SORT :L
IF EMPTYP BF :L [OP :L]
TEST FIRST :L < FIRST BF :L
IFT [OP FPUT (FIRST :L) (SORT (BL :L))]
IFF [OP FPUT (FIRST BF :L) (SORT (FPUT FIRST :L BF BF :L))]
END
```

כאשר נרשום ---> ‏ ?SHOW SORT [35 40 12 27 6 38]

נקבל את הפלט הבא: ‏ [35 12 27 6 38 40]

הפלט הוא רשימה, שבה האיבר בעל הערך הגדול ביותר מוצב בסופה.

<u>הסבר</u>:

בכל מחזור מתבצעת השוואה בין שני נתונים -

```
TEST FIRST :L < FIRST (BF :L)
```

לכן, הביצוע צריך להסתיים כאשר ישאר רק **איבר אחד**, והוא ישמש כנקודת מוצא לערך התחלתי לחישוב רשימת הפלט.

```
IF EMPTYP BF :L [OP :L]
```

את המספר הקטן מבין שני הנתונים שהושוו, יש להציב בראש רשימת הפלט של המחזור הבא. בקריאה לרקורסיה יש לבצע את החישוב על שאר האיברים, פרט לזה שכבר הוצב בפלט. כלומר,

- אם האיבר הראשון הוא שהוצב, אז L: יקבל את הערך 'BF :L'.
- אם האיבר השני הוא שהוצב, אז L: יקבל את כל שאר האיברים **פרט לשני**. לכן, יש לדאוג לכך שהאיבר הראשון יהיה כלול ברשימת קלט זו.

נציג תיאור של הקלט והפלט בכל מחזור:

	הפלט		**הפקודה**
מח' א:	[35 12 27 6 38 40]	<=	FPUT 35 SORT [40 12 27 6 38]
מח' ב:	[12 27 6 38 40]	<=	FPUT 12 SORT [40 27 6 38]
מח' ג:	[27 6 38 40]	<=	FPUT 27 SORT [40 6 38]
מח' ד:	[6 38 40]	<=	FPUT 6 SORT [40 38]
מח' ה:	[38 40]	<=	FPUT 38 SORT [40]

במחזור האחרון הקלט הוא איבר אחד, אשר מוחזר כפלט = [40].

כפי שניתן לראות, הפלט של מחזור א' הוא למעשה הפלט המוחזר מביצוע הפונקציה SORT. עדיין אין זו התשובה שלה אנו מצפים. על-כן, יש להפעיל פונקציה נוספת SORTING, אשר תגרום למיון של כל הרשימה כנדרש.

הגדרת SORTING

```
TO SORTING :LST
IF EMPTYP BF :LST [OP :LST]
OP LPUT (LAST SORT :LST) (SORTING BL SORT :LST)
END
```

כאשר נרשום ---> PR SORTING [35 40 12 27 6 38]

נקבל את הרשימה ממויינת: 6 12 27 35 38 40

הסבר:

בכל מחזור, הפונקציה SORTING מפעילה את ההוראה LPUT

- כדי להציב בסוף רשימת הפלט של המחזור הנקרא:

SORTING BL SORT :LST

- את **האיבר האחרון** של רשימת הפלט של הפונקציה SORT:

LAST SORT :LST

ערך הביטוי 'BL SORT :LST' שבקריאה הרקורסיבית משמש קלט עבור LST: של המחזור הנקרא.

אם נציב את הערך של 'SORT :LST' במקום הביטוי עצמו, נקבל את המשפט הבא עבור המחזור הראשון:

```
OP LPUT
        LAST [35 12 27 6 38 40]
        SORTING BL [35 12 27 6 38 40]
```

הערך של LAST [35 12 27 6 38 40]
הוא '40', והוא יוצב בסוף התוצאה של המחזור הבא.

הערך של BL [35 12 27 6 38 40]
שווה ל-'[35 12 27 6 38]', והוא ישמש ערך ל-LST: של המחזור השני.

במחזור השני כאילו רשמנו:
OP LPUT LAST [35 12 27 6 38] SORTING BL [35 12 27 6 38]

בשלישי: OP LPUT LAST [12 27 6 35] SORTING BL [12 27 6 35]

ולאחריו: OP LPUT LAST [12 6 27] SORTING BL [12 6 27]

ובאחרון: OP LPUT LAST [6 12] SORTING BL [6 12]

כמובן, שאין טעם לבצע את SORT כאשר הרשימה היא בת איבר אחד, ולכן תנאי הסיום הוא:

```
IF EMPTYP BF :LST [OP :LST]
```

וכך, משמשת הרשימה [6] כערך התחלתי לחישוב הפונקציה SORTING.

ניתן לראות שבמשפט הפלט ב-SORTING, מתבצעת הפונקציה SORT פעמיים. אם נצמצם את מספר הקריאות לפעם אחת בלבד, נחסוך בזמן מחשב. ניתן לעשות זאת, אם נקשור את ערך הפונקציה למשתנה שישמש קלט עבור תת-פונקציה SRT, שבה יחושב משפט הפלט הזה.

נשנה אם-כן את SORTING בהתאם:

```
TO SORTING1 :LST
IF EMPTYP BF :LST [OP :LST]
OP SRT (SORT :LST)
END
```

כעת נגדיר את תת-הפונקציה SRT:

```
TO SRT :L
OP LPUT (LAST :L) (SORTING1 BL :L)
END
```

SORTING1 פונה ל-SRT, כאשר L: מקבל את ערך הפלט של 'SORT :LST'. וכך, הפונקציה SORT מתבצעת פעם אחת בכל מחזור. במשפט הפלט ב-SRT מתבצע החישוב על ערך המשתנה L:, ומתבצעת הקריאה הרקורסיבית.

רקורסיה מסוג זה, שבה פונקציה אחת קוראת לשניה, והשניה פונה לראשונה, היא **רקורסיה עקיפה**.

* * * * *

באחד התרגילים בשיעור קודם היה צריך להגדיר פונקציה HAVER, המקבלת שני קלטים, מספר N: ורשימה LST:, ומחזירה כפלט את האיבר המופיע במקום ה-N: ברשימה LST:.
הפונקציה ITEM, אשר קיימת בלוגו משרתת מטרה זו.

הפונקציה ITEM

ITEM מחזירה את האיבר המופיע במקום ה-N: ברשימת הקלט. המספר '1' מצביע על האיבר הראשון ברשימה, '2' - על האיבר השני, וכן הלאה.

המבנה הכללי:	**ITEM מספר רשימה**
כאשר נרשום --->	?PR ITEM 4 [JAN FEB MAR APR MAY]
נקבל:	APR

אם הערך של N: גדול ממספר איברי הרשימה, תופיע ההודעה הבאה:

```
-------------------------------------
TOO FEW ITEMS IN [JAN FEB MAR APR MAY]
-------------------------------------
```

ב-IBM LOGO וב-APPLE LOGO II הפונקציה ITEM פועלת גם על מילה. היא מחזירה את התו המופיע במקום ה-N:.

לדוגמה ---> ?PR ITEM 3 "BEAUTIFUL

הפלט יהיה: A

באחד התרגילים היה צריך להגדיר פונקציה MISPAR, המחשבת את מספר האיברים המופיעים **ברשימה** נתונה. הפונקציה COUNT, אשר נתונה בלוגו מבצעת פעולה זו.

הפונקציה COUNT

COUNT מקבלת רשימה כקלט, ומחזירה כפלט את מספר האיברים הנמצאים ברשימה זו.

כאשר נרשום ---> ?PR COUNT [A [B C [D]] [1 2] XX 3]

נקבל את הפלט הבא ---> 5

ב-IBM LOGO וב-APPLE LOGO II פועלת פונקציה זו גם על מילה. היא מחזירה את מספר התווים המרכיבים את המילה.

אם נרשום ---> ?PR COUNT "PRETTY

נקבל: 6

חיפוש

באחד התרגילים הקודמים היה עלינו להגדיר פונקציה INDEX לחיפוש איבר מסוים ברשימה נתונה. אם האיבר נמצא, הפונקציה מחזירה את מספרו הסידורי ברשימה, ואם הוא לא מופיע - הפונקציה מחזירה את הערך אפס.

בעזרת הפונקציה MEMBERP, ניתן לבדוק אם איבר כלשהו נמצא ברשימה נתונה ולהחליט אם להמשיך בעיבוד, או להפסיק.

חיפוש של איבר ברשימה יכול להתבצע בדרך הפשוטה של שיטת החיפוש הלינארי. בחיפוש זה מתחילים באיבר הראשון ברשימה, עוברים מאיבר לאיבר, עד למציאת האיבר הרצוי. עבור רשימה בת NUM: איברים, פעולת ההשוואה מתבצעת במקרה המקסימלי NUM: פעמים.

כאשר נתונה רשימה לא ממויינת, אין דרך אחרת, אלא לעבור מאיבר לאיבר. לא כן, כאשר הרשימה היא ממויינת, כאן עדיף להשתמש בשיטה של חיפוש בינארי. תיווכח לדעת, שלעתים כדאי מטעמי חסכון בזמן עיבוד, למיין את הרשימה תחילה ואח"כ לערוך את החיפוש.

האלגוריתם של חיפוש בינרי:

א. חשב את מספרו הסידורי של האיבר המופיע באמצע הרשימה, והצב אותו כערך למשתנה IND:.

ב. השווה בין האיבר המבוקש לבין האיבר המופיע במקום ה-IND:.
- אם יש שוויון, הערך של 'IND:' הוא ערך הפונקציה.

ג. אם האיבר הרצוי קטן יותר מהאיבר המופיע במקום ה-IND:, הוא נמצא בתחום של חצי הרשימה שבין האיבר הראשון ועד האיבר ה-IND:. חשב את מספרו הסידורי של האיבר הנמצא באמצעו של תחום זה, והצב אותו כערך חדש ל-IND:, ועבור לשלב ב' של האלגוריתם.

ד. אם האיבר המבוקש גדול יותר - הצב ב-IND: את הערך של המספר הסידורי של האיבר הנמצא באמצע התחום שבין האיבר ה-IND: לבין האיבר האחרון של הרשימה. עבור לשלב ב' של האלגוריתם.

על פי אלגוריתם זה, נעשית ההשוואה הראשונה על האיבר שבאמצע הרשימה. אם אין שוויון עוברים למחצית המתאימה של הרשימה,

ומשווים עם האיבר שבאמצע מחצית זו. ואם שוב אין שוויון, עוברים למחצית המתאימה של המחצית, וכך למעשה, אחרי שתי השוואות בלבד יופחת מספר האיברים הנבדקים לכדי רבע. אחרי 3 השוואות יופחת לשמינית ואחרי 4 נגיע ל-1/16. לפי חישוב זה נמצא, שעבור רשימה בת 2^n-1 איברים דרושות לא יותר מ-n השוואות עד למציאת האיבר הרצוי.

נגדיר אלגוריתם לחיפוש בינרי בשני הליכים, האחד - הליך ראשי HIPUS, והשני - תת-הליך HPS. נשתמש גם בפונקציה MEMBERP, אשר בודקת אם איבר שייך לרשימה.

הגדרת HIPUS

```
TO HIPUS :OBJ :LST
TEST MEMBERP :OBJ :LST
IFT [OP HPS 1 (INT (COUNT :LST) / 2) (COUNT :LST)]
IFF [OP (SE :OBJ [IS NOT MEMBER OF] :LST)]
END
```

ההליך HIPUS מקבל במשתנה OBJ: את האיבר המבוקש, ובמשתנה LST: הוא מקבל רשימה ממויינת. תחילה ההליך בודק, באמצעות MEMBERP, אם OBJ: נמצא ברשימה LST:.

א. אם כן ('IFT'), הוא פונה לפונקציה HPS ומעביר לה 3 ערכים:
- ערך הגבול התחתון, שהוא המספר הסידורי של האיבר הראשון ברשימה.
- המספר הסידורי של האיבר המופיע באמצע הרשימה: (INT (COUNT :LST) / 2)
- ערך הגבול העליון, שהוא המספר הסידורי של האיבר האחרון ברשימה: (COUNT :LST)

ב. אם לא ('IFF'), הוא מודיע שהערך של OBJ: איננו שייך לרשימה LST:.

הגדרת HPS

```
TO HPS :LWR :IND :HGR
IF :OBJ = ITEM :IND :LST [OP :IND]
TEST :OBJ < ITEM :IND :LST
IFT [OP HPS :LWR (INT (:LWR + :IND) / 2) :IND]
IFF [OP HPS :IND (INT (:IND + :HGR) / 2) :HGR]
END
```

בהליך זה, המשתנה LWR: מקבל את ערך הגבול התחתון של התחום הנבדק, המשתנה IND: מקבל את המספר הסידורי של אמצע התחום הנבדק, והמשתנה HGR: מקבל את ערך הגבול העליון של אותו תחום.
בתחילת החישוב, אם יש שוויון מוחזר הערך של IND:.
אחרת, בודקים אם OBJ: קטן יותר מהאיבר הנמצא במקום ה-IND: -

```
TEST :OBJ < ITEM :IND :LST
```

בשל סימן היחס '>', פונקציה זו מתאימה רק לחיפוש של נתון מספרי ברשימת מספרים. אם רוצים להגדיר פונקציה עבור חיפוש מילה ברשימת מילים, יש להשתמש בפונקציה המשווה מילים לפי הסדר המילוני, כמו למשל BEFOREP.

בהתאם לתשובה של משפט ה-TEST, נעשית פניה רקורסיבית ל-HPS עם העברת הערכים של המחצית המתאימה של הרשימה, או של קטע ממנה.

<u>שים לב:</u> מאחר ש-OBJ: ו-LST: הם משתנים לא-מקומיים, הם מוכרים על-ידי HPS, ולכן ניתן לערוך בהם חישוב בתוך ההגדרה של הליך זה.

שיטה למיון מהיר

קיימות שיטות מיון מהירות יותר. אחת מהן, שנכנה אותה TURBOSORT, פועלת בדרך הבאה:

נתונה רשימה L: שאינה ממויינת. משמאל **לאיבר הראשון** שלה מציבים בתת-רשימה LFL: את כל האיברים הקטנים ממנו, ומימינו מציבים בתת-רשימה RTL: את שאר האיברים. מאחדים את הכל ברשימה אחת באמצעות הפונקציה SE. חוזרים על תישוב זה לכל אחת מתת-הרשימות המתקבלות, כל עוד יש בהן יותר מאיבר אחד.

לדוגמה, נתונה הרשימה: [34 8 23 50 14 -3 47 50 66 -10 31]

במחזור הראשון נקבל רשימה, שהאיבר הראשון
בה הוא תת-רשימה: [8 23 14 -3 -10 31]
האיבר השני הוא המספר הראשון ברשימה המקורית: 34
האיבר האחרון הוא תת-רשימה: [50 47 50 66]

נאחד את האיברים ונקבל (רשימה, מספר, רשימה):

[[8 23 14 -3 -10 31] 34 [50 47 50 66]]

במחזור 2 נקבל:

[[-3 -10] 8 [23 14 31]] ↓ [[47] 50 [50 66]]

במחזור 3 נקבל:

[[-10] -3] ↓ [[14] 23 [31]] ↓ [47] ↓ [50 [66]]

נמשיך ונקבל:

[-10] ↓ ↓ [14] ↓ [31] ↓ ↓ ↓ ↓ [66]

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

הפלט הסופי: [-10 -3 8 14 23 31 34 47 50 50 66]

במחזור הראשון פעלנו על רשימה אחת. במחזור 2 אנו פועלים על שתי רשימות, הנמצאות משני הצדדים של המספר 34.
במחזור השלישי לפנינו רשימות רבות יותר, אשר יש לטפל בכל אחת מהן בנפרד. אח"כ מקבלים מספר רשימות בנות איבר אחד.

מחלצים את האיברים מן הרשימות ומקבלים איברים בודדים בלבד, המהווים את הפלט של ההליך.

הרשימה הממויינת מתקבלת על-ידי איחוד של כל האיברים המתקבלים בכל מחזור.

לצורך החישוב נגדיר שתי פונקציות, האחת ראשית QSRT, והשניה - תת-פונקציה PIZUL. QSRT פונה ל-PIZUL, אשר מחזירה כפלט רשימה המורכבת משתי תת-רשימות: בראשונה נמצאים כל האיברים הקטנים מהאיבר הראשון, ובשניה - שאר האיברים שאינם קטנים ממנו. QSRT מאחדת באמצעות הפונקציה SE את שתי תת-הרשימות האלו כשביניהם מוצב האיבר הראשון.

בפניה הרקורסיבית נוכל להשתמש במשפט הבא:

OP (SE **QSRT** (<u>FIRST PIZUL :L</u>) FIRST :L **QSRT** (<u>LAST PIZUL :L</u>)

* * * * *

תרגילים (המשך)

18. בתרגיל 26 בפרק הקודם צריך היה להגדיר פונקציה LIFNEIP לבדיקה, אם מילה נתונה מופיעה לפני מילה אחרת על פי הסדר המילוני. השתמש בפונקציה זו על מנת להגדיר תכנית מיון MIUN, למיון רשימה של מילים.

19. נתונות שתי רשימות **לא** ממויינות של מילים. הגדר פונקציה המחזירה רשימה אחת ממויינת, שאיבריה הם האיברים של שתי הרשימות.

20. הגדר פונקציה MIZUG, המקבלת שתי רשימות ממויינות של שמות של סופרים, ומאחדת אותם לרשימה ממויינת אחת.

21. הגדר פונקציה המקבלת שתי רשימות. האחת STUDS: בה מופיעים שמות של תלמידים, והשניה GRADES: ובה ממוצע הציונים שלהם בהתאמה. נוסף לשתי הרשימות מקבלת הפונקציה שם של תלמיד

:NAME ומחזירה את הציון שלו מרשימת הציונים.

לדוגמה,

- אם :STUDS = [AAA BBB CCC DDD EEE FFF GGG]

- ו :GRADES = [60 75 90 70 85 80 95]

- ו :NAME = EEE

אז הפלט יהיה: 85.

22. הגדר פונקציה המקבלת שתי רשימות. האחת - רשימת חודשי השנה, והשניה - כמות המשקעים של כל חודש בהתאמה. על הפונקציה להחזיר את החודש שבו היתה כמות המשקעים הרבה ביותר.

23. בתרגיל מס' 8 בפרק 8 (פונקציות נתונות), היה עליך לכתוב תכנית המחשבת איזה יום בשבוע חל NUM: ימים אחרי יום DAY:, ומדפיסה את שם היום במילים.
באמצעות הפונקציה ITEM, הגדר פונקציה המורכבת מהוראה אחת בלבד לפתרון אותה בעיה. דהיינו, על הפונקציה להחזיר את שם היום במילים.

24. נתונות שתי רשימות: TLM: היא רשימה ממויינת של שמות תלמידים (הנח שאין שני שמות זהים); TZT: היא רשימה של מספרי תעודת הזהות שלהם בהתאמה. הגדר פונקציה למציאת מספר תעודת הזהות של התלמיד ששמו ניתן ב-NAME:.

25. הגדר תכנית המבצעת מיון עבור רשימה נתונה RESH: בדרך הבאה:

- מצא את האיבר הגדול ביותר באמצעות פונקציה BIGST: והצב אותו בסוף רשימת הפלט של המחזור הבא.
- מחק באמצעות פונקציה SELECT את האיבר הזה מ-RESH: של המחזור הנוכחי, והעבר את הרשימה החדשה כקלט ל-RESH: של המחזור הבא.

26. כתוב תכנית למיון רשימת מספרים על פי השיטה TURBOSORT שתוארה בעמוד הקודם.

פרק 13

רשימות מקוננות

איבר של רשימה יכול להיות תו, מילה, מספר או רשימה. רשימה המוכלת בתוך רשימה היא **רשימה פנימית**. רשימה פנימית יכולה גם היא להכיל איברים שהם רשימות, ולא רק מילים, או מספרים. מכאן, שרשימות יכולות להכיל רשימות של רשימות בכל רמה שהיא. רשימה מסוג זה היא **רשימה מקוננת** (Nested List).

את ה"רמה" של הרשימה הראשית נגדיר בדרך זו:

- רשימה שאין בה אף לא איבר אחד שהינו רשימה, היא רשימה בת **רמה אחת**. כלומר, היא רשימה של **אטומים**, שאיבריה הם מילים או מספרים.
- רשימה בת **שתי רמות**, היא רשימה שיש בה **לפחות** איבר אחד שהוא רשימה פנימית. רשימה פנימית זו הינה בת רמה אחת בלבד.
- רשימה בת **שלוש רמות**, היא רשימה שיש בה **לפחות** איבר אחד שהוא רשימה בת שתי רמות.

ובאופן כללי - רשימה בת **N** רמות, היא רשימה שיש בה איבר אחד **לפחות** שהוא רשימה בת **N-1** רמות. רשימה בת N רמות הינה רשימה מקוננת.

איחוד של רשימות

הפונקציה SENTENCE (או SE) מאחדת רשימות, כלומר מצרפת **איברי** רשימה אחת עם **איברי** רשימה אחרת ויוצרת רשימה אחת. תכונה זו של הפונקציה SE מאפשרת להגדיר פונקציה SQUASH, המקבלת רשימה של רשימות בכל רמה שהיא, ומחזירה רשימה בת רמה אחת. וכך, איבריה יהיו כל **האטומים** שהופיעו ברשימת הקלט.

הגדרת SQUASH

```
TO SQUASH :LST
IF EMPTYP :LST [OP []]
TEST LISTP (FIRST :LST)
IFT [OP SQUASH SE (FIRST :LST) (BF :LST)]
OP FPUT (FIRST :LST) (SQUASH BF :LST)
END
```

נרשום --->

```
?SHOW SQUASH [[BB 1 [DDD 22] [F [PI]] 3.14] [I] J]
```

ונקבל את הרשימה הבאה:

```
[BB 1 DDD 22 F PI 3.14 I J]
```

כל האיברים שאינם רשימות, אלא מילים או מספרים בכל רמה שהיא, הופיעו כאיברים של רשימת הפלט ברמה אחת. רשימת הקלט LST: יכולה להכיל איברים שהם בעצמם רשימות. לכן יש לבדוק כל אחד מן האיברים, ולקבוע אם הוא רשימה:

```
TEST LISTP (FIRST :LST)
```

- אם לא, כלומר אם האיבר הנבדק הינו מילה או מספר, יש להציב אותו **בתחילת** רשימת הפלט של המחזור הבא:

```
OP FPUT (FIRST :LST) (SQUASH BF :LST)
```

- אבל, אם הוא כן רשימה, יש לאחד את **איבריו** עם שאר איברי LST: באמצעות הפונקציה SE לרשימה אחת, שתהווה קלט למחזור הבא:

```
IFT [OP SQUASH SE (FIRST :LST) (BF :LST)
```

מגיעים למחזור האחרון כאשר אין יותר איברים ב-LST:. הרשימה הריקה משמשת כערך התחלתי (נקודת מוצא) לחישוב הפלט:

```
IF EMPTYP :L [OP []]
```

נציג איפוא את התיאור של הקלט והפלט בכל מחזור:

	הקלט	הפלט	
א.	[BB 1 [DDD 22] [F [PI]] 3.14] [I] J]	פלט מחזור ב'	
ב.	[BB 1 [DDD 22] [F [PI]] 3.14 [I] J]	פלט מחזור ג'	FPUT BB
ג.	[1 [DDD 22] [F [PI]] 3.14 [I] J]	פלט מחזור ד'	FPUT 1
ד.	[[DDD 22] [F [PI]] 3.14 [I] J]	פלט מחזור ה'	
ה.	[DDD 22 [F [PI]] 3.14 [I] J]	פלט מחזור ו'	FPUT DDD
ו.	[22 [F [PI]] 3.14 [I] J]	פלט מחזור ז'	FPUT 22
ז.	[[F [PI]] 3.14 [I] J]	פלט מחזור ח'	
ח.	[F [PI] 3.14 [I] J]	פלט מחזור ט'	FPUT F
ט.	[[PI] 3.14 [I] J]	פלט מחזור י'	
י.	[PI 3.14 [I] J]	פלט מחזור יא'	FPUT PI
יא.	[3.14 [I] J]	פלט מחזור יב'	FPUT 3.14
יב.	[[I] J]	פלט מחזור יג'	
יג.	[I J]	פלט מחזור יד'	FPUT I
יד.	[J]	פלט מחזור טו'	FPUT J
טו.	[]	[]	

ניתן לראות ש-FPUT פועלת רק כאשר האיבר הראשון של כל מחזור אינו רשימה. כאשר האיבר הראשון הוא רשימה, מאחדת SE את איבריו עם שאר איברי הרשימה של אותו מחזור לרשימה אחת, שתשמש קלט למחזור הבא.

לדוגמה, רשימת הקלט של מחזור ה' היא:

[DDD 22 [F [PI]] 3.14 [I] J]

האיבר הראשון ברשימה זו הוא 'DDD', שאינו רשימה. לכן, במחזור זה מציבה הפונקציה **FPUT** את DDD בתחילת רשימת הפלט של המחזור הנקרא על ידה, כלומר מחזור ו'. מחזור ו' מקבל כקלט את הרשימה **ללא** האיבר הראשון.

דוגמה נוספת - רשימת הקלט של מחזור ז' היא:

[[F [PI]] 3.14 [I] J]

האיבר הראשון כאן הוא רשימה [F [PI]]. על מנת להגיע לאטומים, מאחדת SE את איברי **האיבר הראשון** שהם האיבר F והאיבר [PI] עם שאר איברי הרשימה (J, [I] ו-3.14) לרשימה אחת ומעבירה אותה כקלט למחזור הבא שהוא מחזור ח'. במקרה זה, רשימת הפלט של מחזור

ז' תהיה רשימת הפלט של המחזור אשר נקרא על ידו, דהיינו, של מחזור ח'.

* * * * *

תרגילים

1. הגדר פרדיקט הבודק אם שתי רשימות הן בעלות אורך שווה.

2. בדו"ח הציונים של התלמידים, רשומים בכל שורה 5 נתונים. הנתון הראשון הוא שם התלמיד, וארבעת האחרים מהווים את ציוניו במקצועות מתמטיקה, אנגלית, מחשבים ועברית. כל איבר ברשימת הקלט הינו למעשה רשימה של תלמיד אחד, שבה מופיע שמו כאיבר ראשון, ועוד ארבעת הציונים שלו. הגדר פונקציה המקבלת את רשימת התלמידים וציוניהם, ומחזירה את **רשימת השמות** של כל התלמידים שברשימה.

לדוגמה, אם הקלט הוא:
[[NIR 80 85 87 85] [OREN 85 75 95 80] [DAN 90 80 92 90]]
אז הפלט יהיה: [NIR OREN DAN]

3. הגדר פונקציה המקבלת רשימה לפי המתואר בתרגיל 2, ומחזירה את רשימת הציונים של המקצוע המופיע במקום ה-N:.

על פי הדוגמה שבתרגיל 2, רשימת הציונים של כל התלמידים במקצוע המופיע **כאיבר רביעי** ברשימת כל תלמיד היא:
[87 95 92]

4. כתוב תכנית המקבלת תאריך כלשהו, ומחשבת כמה ימים חלפו מתחילת אותה שנה ועד לאותו תאריך ("כולל"). זכור להתחשב במספר הימים בכל חודש.

5. כתוב תכנית המקבלת שני תאריכים ומחשבת כמה ימים יש ביניהם. לדוגמה, ההפרש בין ה-9 בינואר לבין ה-5 בינואר הוא 4 ימים.

6. כתוב תכנית המחקה 50 זריקות של קוביה, ופולטת כמה הגרלות היו לכל מספר.

7. הגדר תכנית המקבלת רשימת הודעות NEWS: כדוגמת "עיקר החדשות", הודעות פרסומת, או מאמר כלשהו. הפלט הוא הצגה של הרשימה הזו שוב ושוב בשורה אחת ויחידה על המסך בתנועה מימין לשמאל, כפי שרואים בפרסומת אלקטרונית ("פרסומת נעה").

8. הגדר תכנית המקבלת כל מילה שהיא, ומדפיסה אותה לפי הדוגמה הבאה. עבור המילה KUKURIKU נקבל:

```
      K
     K U
    K U K
    . . . .
 K U K U R I K U
```

שים לב, שהתחלנו מן האות הראשונה.

9. הגדר הליך המקבל מילה המורכבת ממספר תווים אי-זוגי, ומציגה את הפלט לפי תבנית נתונה. אם למשל, נתונה המילה JERUSALEM, אז הפלט יהיה:

```
      S
    U S A
   .........
J E R U S A L E M
```

שים לב, שהתחלנו מן האות האמצעית.

10. הגדר פונקציה SETATOM המקבלת רשימה בכל רמה שהיא, ומחזירה את רשימת כל האטומים (מילים או מספרים) כקבוצה, ללא חזרות.

11. הגדר פונקציה הבוחרת מספר אקראי R: בתחום (6 .. 1), ומחזירה אותו ברשימה בת R: רמות, כשהוא מופיע ברשימה הפנימית ביותר.

לדוגמה: אם המספר הנבחר הוא 3, הפלט יהיה: [[[3]]]. אם המספר הוא 1, הפלט יהיה: [1].

* * * * *

חישובים מתקדמים במספרים

קיימות שיטות ספירה אחרות מלבד השיטה העשרונית. בשיטה העשרונית הבסיס הוא המספר 10, והשימוש הוא ב-10 ספרות שהראשונה בהן היא אפס, דהיינו: 9 ... 2 1 0.

כאשר הצגנו את קוד ASCII אמרנו, שמספרי הקוד הם מ-0 ועד 255. אין זו קביעה שרירותית. כל תו מיוצג במחשב על-ידי **8 סיביות** שהן **בית** אחד (BYTE). כל **סיבית** (BIT) יכולה להיות באחד משני מצבים, דלוק (ON) או כבוי (OFF), מצבים אלה נוכל לתאר על-ידי שתי הספרות '1' או '0' בהתאמה. לכן נמצא, **שמספר הצירופים השונים** שנוכל לייצג בבית אחד בן 8 סיביות הוא: $256 = 2^8$.

כאשר למדנו את ההוראה REPEAT נאמר, שבגירסה של APPLE LOGO מספר הפעמים המקסימלי שאפשר לרשום בהוראה REPEAT הוא 65535 (ב-IBM LOGO התגברו על מגבלה זו). למניית מספר הפעמים של ביצוע ההוראה משתמשים בשני בתים, שהם 16 סיביות. באמצעותם נוכל לייצג 2^{16}, 65536 צירופים, מ-0 ועד 65535. על-כן, הערך המספרי הגדול ביותר שניתן לבטא בדרך זו הוא 65535.

בשיטה ה-**n**-ית, שבה הבסיס הוא **n**, יהיו **n** ספרות שונות שהראשונה בהן היא '0', והאחרונה תהיה **'n-1'**.

ערכה של הסיפרה מחושב על-ידי הכפלת הסיפרה בחזקה המתאימה של הבסיס n, בהתאם למקומה של הסיפרה במספר. עבור סיפרת היחידות החזקה המתאימה היא '0', עבור הסיפרה השניה מימין החזקה היא '1', עבור הסיפרה השלישית מימין החזקה היא '2' וכן הלאה. ערך המספר עצמו הינו **סכום** כל המכפלות של כל הספרות.

זכור, כי כל מספר בחזקת '0' שווה ל-1.

בשיטה הבינרית (Binary), בבסיס 2, יש שתי ספרות בלבד: 0 ו-1.
כאשר נחשב את ערך המספר 1101001_2 נקבל:

$$1*2^6 + 1*2^5 + 1*2^3 + 1*2^0 = 105_{10}$$

בשיטה האוקטלית (Octal), הבסיס הוא 8 ויש בה 8 ספרות: '0 1...7'.
אם נתון המספר 275 בבסיס 8, נציין אותו כך: 275_8. ערכו בשיטה העשרונית יהיה לפי החישוב הבא:

$$2*8^2 + 7*8^1 + 5*8^0 = 189_{10}$$

בשיטה ההקסדצימלית (Hexadecimal), בבסיס 16, יש 16 סמלים שונים. הספרות: '0 1...9', והאותיות: 'A B C D E F'. האות A מציינת את הערך 10, B=11, C=12, D=13, E=14 ו-F=15.
ערך המספר $20E9_{16}$ יהיה:

$$2*16^3 + 0*16^2 + E*16^1 + 9*16^0 = 4329_{10}$$

עד כאן הפכנו מספר בבסיס n למספר בבסיס 10.

הסבת מספר מבסיס עשרוני למספר בבסיס n כלשהו, נעשית על-ידי חלוקת המספר בערך הבסיס n בפעולות חילוק חוזרות, עד שמקבלים מנה של אפס. את המספר החדש בונים על-ידי רישום השאריות של פעולות החילוק מימין לשמאל.

דוגמה: הסבת המספר 105_{10} למספר בינרי.

```
105 / 2 = 52 (1)  ──────────┐
 52 / 2 = 26 (0)  ────────┐ │
 26 / 2 = 13 (0)  ──────┐ │ │
 13 / 2 =  6 (1)  ────┐ │ │ │
  6 / 2 =  3 (0)  ──┐ │ │ │ │
  3 / 2 =  1 (1)  ┐ │ │ │ │ │
  1 / 2 =  0 (1) ┐│ │ │ │ │ │
                 ↓↓ ↓ ↓ ↓ ↓ ↓
                 1 1 0 1 0 0 1
```

מכאן ש- $105_{10} = 1101001_2$

ההסבה מבסיס עשרוני לבסיס אוקטלי, לבסיס הקסדצימלי, או לכל בסיס n שהוא, דומה להסבה מבסיס עשרוני לבינרי בהבדל אחד: כאן המחלק הוא 8 או 16 או כל ערך n, אשר מציין את הבסיס שאליו מסיבים את המספר.

נציג לדוגמה 2 פונקציות להמרה של מספר מייצוג בשיטת בסיס אחד לייצוג בשיטת בסיס אחרת.

הגדרת BIN.TO.DEC

פונקציה ההופכת מספר בינרי למספר עשרוני.

```
TO BIN.TO.DEC :NUM :HZK
IF EMPTYP :NUM [OP 0]
OP (:HZK * LAST :NUM) + (BIN.TO.DEC BL :NUM (:HZK * 2))
END
```

הפרמטר הראשון הוא המספר המיועד להסבה והפרמטר השני הוא ערך הבסיס בחזקת '0' (אפס), עבור השלב הראשון של החישוב.

כאשר נרשום --->

```
?PR BIN.TO.DEC 11011 1
```

נקבל את התוצאה הבאה:

```
27
```

הסבר:

בתחילת הביצוע הערך של 'LAST :NUM' הוא סיפרת היחידות, אשר מוכפלת בבסיס, אשר מועלה בחזקת '0'. לכן, בקריאה למחזור הראשון קיבל המשתנה HZK: את הערך '1'. נחבר תוצאה זו לשאר התוצאות של המכפלות שיחושבו בהמשך לכל אחת מהספרות, כאשר בכל מחזור המשתנה HZK: מקבל ערך השווה לערך הבסיס כשהוא מועלה בחזקה הבאה.

הגדרת DEC.TO.BIN

פונקציה להפיכת מספר עשרוני למספר בינרי.

```
TO DEC.TO.BIN :MIS
IF :MIS < 2 [OP :MIS]
OP WORD (DEC.TO.BIN INT :MIS / 2) (REMAINDER :MIS 2)
END
```

הסבר:

נרשום את השאריות של החלוקה מימין לשמאל, כדי לבנות את המספר שמהווה רצף של ספרות 0 ו-1. זאת עושים באמצעות הפונקציה WORD, שמצרפת את השאריות שיחושבו בהמשך,

DEC.TO.BIN INT :MIS / 2

עם השארית שחושבה באותו מחזור: REMAINDER :MIS 2

תנאי הסיום הוא כאשר המנה היא '0': IF :MIS < 2

נקודת המוצא היא השארית האחרונה: [OP :MIS]

* * * * *

תרגילים (המשך)

12. הגדר פונקציה להסבה של מספר אוקטלי (בסיס 8) למספר עשרוני.

13. הגדר פונקציה להסבה של מספר עשרוני למספר אוקטלי.

14. הגדר פונקציה להסבה של מספר הקסדצימלי (בסיס 16) למספר עשרוני. כאשר הערך מיוצג על-ידי אות ולא על-ידי סיפרה, יש לתרגם את האות לערכה המספרי לצורך חישוב המכפלה.

15. הגדר פונקציה להסבה של מספר עשרוני למספר הקסדצימלי. כאשר השארית גדולה מ-9 יש להפוך אותה לאות המייצגת את ערכה.

16. הגדר פונקציה להסבת מספרים מכל בסיס שהוא למספר עשרוני, ופונקציה אחרת להסבת מספרים עשרוניים לכל בסיס אחר.

17. הגדר פונקציה להסבת מספרים מכל בסיס שהוא לכל בסיס אחר.

הדרכה: יש להפוך את המספר לבסיס 10 תחילה, ואת התוצאה להסב לבסיס הרצוי.

* * * * *

יצוג מספרים בשיטה המדעית

בחישובים אריתמטיים אפשר לקבל מספרים גדולים אשר עוברים את גבול הדיוק כפי שהוא מוגדר בשפה. כדי להמנע מתקלה זו, מציגים מספרים אלה בשיטה המדעית באמצעות הנקודה הצפה.

ב-IBM LOGO המספר השלם הגבוה ביותר שניתן להציג מורכב מ-10 ספרות, וערכו 9999999999.

ב-APPLE LOGO המספר השלם הגבוה ביותר שניתן להציג הוא המספר 2147483647. הערך של המספר השלם הגבוה ביותר נקבע על פי השימוש בארבעה בתים, כלומר '4*8' סיביות. ב-32 סיביות ניתן לבטא 2^{32} צירופים שונים מ-'0' ועד '2147483647'.

אם נרשום ---> ?PR REMAINDER 76599554472 1000

נקבל בגירסת APPLE LOGO הודעת שגיאה, כי פונקציה זו פועלת רק על מספרים בדיוק מלא.

```
--------------------
NUMBER TOO BIG
--------------------
```

ב-IBM LOGO נקבל את התוצאה:

470

שים לב, כי התוצאה המדויקת הינה 472 **ולא** 470.

אם המספר הוא מעל ל-10 ספרות (הספירה נעשית משמאל לימין, בדומה ל-ITEM), לוגו תעגל אותו תחילה ל-10 ספרות, ורק אחר-כך היא תחשב את השארית של המספר המעוגל שהתקבל. מימין לשארית היא תוסיף אפסים, כמספר הספרות הימניות שנשמטו מן המספר המקורי. עיגול המספר ל-10 ספרות גורם לכך שאין הוא מוצג **בדיוק** על פי ערכו.

השימוש בשיטה המדעית מאפשר לאחסן מספרים גדולים מאוד ללא צורך בניצול מספר רב של בתים. לעומת זאת, מאבד המספר מדיוקו, לפי מספר הבתים שנקבעו לאחסנה שלו.

למשל, הערך של הביטוי 1/3

מאוחסן כך לפי השיטה המדעית: 0.3333333333

כל תלמיד יודע ש-3 פעמים 1/3 שווה 1, אך לא כן במחשב. המכפלה של ...0.333 ב-3 לא תתן את הערך 1.

לדוגמה, נגדיר הליך המקבל נתון מספרי כלשהו וכופל אותו ב-3:

```
TO KAFUL3 :MIS
PR 3 * :MIS
END
```

כאשר הנתון הוא 1/3 נרשום --->

```
?KAFUL3 1/3
0.9999999999
```

נקבל:

בשיטה המדעית בגירסה של IBM LOGO, מוצג המספר על-ידי חזקה של בסיס 10, המוכפלת במקדם. המקדם מיוצג על-ידי מספר עשרוני שערכו בין 1 ל-9.

לדוגמה:

המספר 34256746543 מוצג על-ידי המחשב כך: 3.425674654E+0011

והמשמעות היא: $3.425674654 * 10^{11}$

מעריך החזקה (Exponent), הינו המספר המופיע מימין לאות E. האות E נמצאת במקום העשירי מהנקודה הצפה ואחריה מספר מורכב מארבע ספרות. הוא מציין את החזקה '+0011', סימן הפלוס מציין שהחזקה חיובית.

מימין לסיפרה 3 במספר המקורי יש 11 ספרות ולכן מעריך החזקה הוא 11. ולדוגמה, את המספר 849.76 נציג כך: 8.497600000E0002.

במספרים שליליים מופיע הסימן מינוס '-' בתחילת המספר.

עבור מספרים בשבר עשרוני הקטנים מ-1, שהחזקה בהם היא שלילית, מופיע סימן מינוס אחרי האות E.

בשיטה זו מורכב המספר מ-17 תווים, כאשר התו הראשון יכול להיות סיפרה, או סימן המינוס '-'.

כאשר החזקה השלילית היא '-1', מספר הספרות אחרי הנקודה העשרונית הוא 10, בעיגול הסיפרה העשירית בהתאם לצורך.

לדוגמה,

המספר 0.00001234567 מוצג כך: 1.234567000E-0005

והמשמעות היא: $1.234567 * 10^{-5}$

תרגיל: ערוך בדיקה וגלה בעצמך כיצד מוצגים מספרים בשיטה המדעית בגירסה שבידך.

קליטת תווים מן המקלדת

פונקציית הקלט READCHAR (RC)

פונקציה זו משמשת לקליטת ערך מן המשתמש תוך כדי ביצוע ההוראה בה היא רשומה. ערך זה מתקבל באופן הבא: לוגו מפסיקה את פעולתה וממתינה עד אשר המשתמש יקיש על מקש כלשהו. היא מזהה את המקש הנלחץ ומעבירה את הערך שלו כקלט להוראה הפועלת על RC.

לדוגמה: נרשום ---> ?PR WORD RC "ALL

כל עוד לא נלחץ מקש כלשהו, RC אינה קולטת כל ערך שהוא, ההוראה אינה יכולה להתבצע ותכנית לוגו תמשיך להמתין. ברגע שנקיש על מקש כלשהו, למשל על האות 'B', מועבר ערך זה באמצעות RC כנתון אל הפונקציה WORD, וכך נקבל: BALL

הפונקציה WORD צירפה את התו 'B' שהקשנו אל התווים 'ALL'.

באחד התרגילים הקודמים צריך היה להגדיר תכנית המקבלת מספר ומדפיסה אותו ואת סכום ספרותיו. האם ברור לך עתה מדוע הגבלנו את מספר הספרות של הקלט ל-9.

נכתוב כעת תכנית לפתרון התרגיל עבור קלט של מספר בכל גודל שהוא. על מנת להתגבר על הבעיה של הגבלת מספר הספרות במספר המוצג שלא בשיטה המדעית, נגדיר פונקציה שתקלוט את הספרות מהמשתמש בזו אחר זו באמצעות הפונקציה RC, ותחבר אותן למילה אחת.

הגדרת PUTLIST

```
TO PUTLIST :DIGIT
IF NOT NUMBERP :DIGIT [OP " ]
OP WORD :DIGIT PUTLIST RC
END
```

בקריאה לפונקציה נרשום ---> ?PR PUTLIST RC

לוגו תמתין להקשת מקש כלשהו על-ידי המשתמש. הפונקציה PUTLIST מבצעת שתי פעולות על ערך המקש (התו) ש-RC מספקת לה. ערך זה נקשר למשתנה DIGIT:, שבאמצעותו מתבצעים החישובים.

הפונקציה PUTLIST בודקת בכל מחזור

אם הקלט הוא סיפרה: IF NOT NUMBERP :DIGIT [OP "]

אם הקלט הוא סיפרה, כלומר אם התנאי לא מתקיים, היא מחברת אותו עם ערך הפונקציה של המחזור הבא למילה:

```
OP WORD :DIGIT PUTLIST RC
```

בכל מחזור מופעלת RC, ועל כן בכל פניה ל-PUTLIST לוגו תמתין לקליטת תו קלט נוסף. כתנאי לסיום משמש תו קלט שאיננו סיפרה. בסיום ביצוע התכנית נקבל מילה המורכבת מספרות בלבד.

התכנית שלפניך קולטת מספר ומדפיסה אותו ואת סכום ספרותיו. PUTLIST מבצעת את קליטת המספר. לחישוב סכום הספרות נפנה לפונקציה ADD המקבלת רשימת מספרים, או מילה המורכבת מספרות, ומחשבת את סכום איבריה.

הגדרת ADD

```
TO ADD :MIS
IF EMPTYP :MIS [OP 0]
OP (FIRST :MIS) + ADD BF :MIS
END
```

בקריאה ל-ADD נרשום ---> ?PR ADD PUTLIST RC

הערך של הפונקציה PUTLIST הוא מילה המורכבת מספרות, והוא מועבר למשתנה MIS:. הפונקציה ADD מחזירה את סכום הספרות המרכיבות מילה זו.

בפתרון התרגיל **לא** נוכל לרשום את המשפט הבא: ?PR SE PUTLIST RC (ADD PUTLIST RC)

הסיבה לכך היא, ש-PUTLIST תתבצע פעמיים, ויהיה על המשתמש להקיש פעמיים את אותו המספר. במקרה זה עלול להיות מצב, שהמילה שתוקש בפעם הראשונה ותוצג, תהיה שונה מן המילה שתוקש בפעם השניה ותחושב. כך נוכל לקבל מספר שסכום ספרותיו איננו זהה עם הסכום שנרשם על המסך.

כדי ש-PUTLIST תתבצע פעם אחת בלבד, נגדיר פונקציה PELET עם משתנה WRD:, שאליו נקשור את ערך הפונקציה PUTLIST. PELET תציג את WRD: ואת ערך הפונקציה ADD, אשר תפעל על הערך המועבר לה באמצעות המשתנה WRD:.

הגדרת PELET

```
TO PELET :WRD
OP SE :WRD (ADD :WRD)
END
```

כעת נוכל לרשום ---> `?PR PELET PUTLIST RC`

ונקבל את הקלט שהוזן על-ידי המשתמש.

כאשר מריצים תכנית שדורשת קלט מן המשתמש, רצוי שתהיה לו הנחיה בדבר הקלט הנדרש ממנו. לשם כך נגדיר הליך PRNUM, אשר מציג על המסך הודעה מתאימה ואחר-כך פונה לפונקציה PELET.

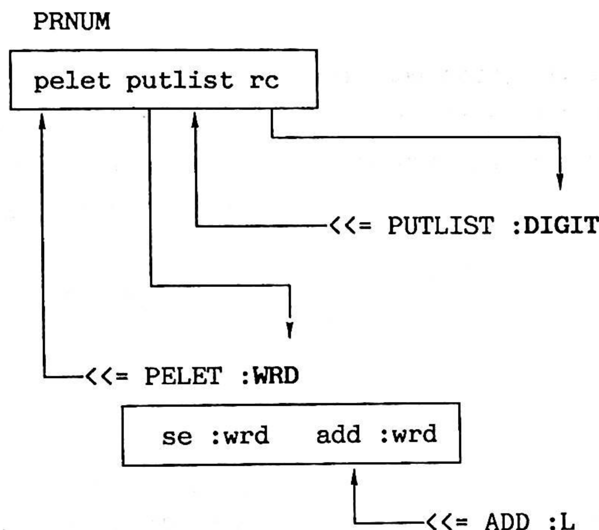
הגדרת PRNUM

```
TO PRNUM
CLEARTEXT
PR [INPUT AN INTEGER NUMBER AND THEN PRESS RETURN]
PR [] PR []
PR PELET PUTLIST RC
END
```

במקרה זה די לרשום ---> `?PRNUM`

תכנית זו היא אחת הדוגמאות של תכנות מבני, המתבצעת באופן פונקציונלי.

נוכל לתאר זאת באמצעות התרשים הבא:



ניתן לראות על פי התרשים, ש-PRNUM פונה לשתי פונקציות PUTLIST ו-PELET, כחלק מהוראות החישוב שלה. את PUTLIST היא מפעילה על מנת שהפלט שלה ישמש קלט לפונקציה PELET, והפלט של PELET הוא תוצאת החישוב של PRNUM.

הפונקציה PUTLIST פועלת על RC. הערך שמתקבל מ-RC משמש קלט עבור המשתנה DIGIT:. ותוצאת החישוב של PUTLIST כאמור, מהווה קלט עבור המשתנה WRD: המוגדר בפונקציה PELET.

הפונקציה PELET מאחדת באמצעות SE את הערך של WRD: עם ערך הפונקציה ADD. איחוד זה הינו הערך שהיא מחזירה ל-PRNUM.

בהוראות שאינן מתמטיות, כאשר מספר מוגדר כמילה, שפת לוגו מתייחסת אליו כאל רצף של תווים ולא לפי ערכו המספרי. היא מדפיסה אותו כפי שהוא, ולא לפי השיטה המדעית, גם אם הוא עובר את גבול הדיוק.

בדוגמה זו נוכל לפנות ל-PELET כשערך הקלט מלווה בגרשיים: והתוצאה תהיה:

```
?PR PELET "776463968553
776463968553 69
```

פעולות חשבון בבסיס n

פעולות החשבון בבסיס n דומות לאלו אשר נעשות בבסיס עשרוני. ההבדל הוא בתחום המספרים שמשתמשים בהם בכל שיטה.

בחיבור שני מספרים מתבצעות הפעולות מימין לשמאל. תחילה מחברים את סיפרת האחדות של כל אחד מהמספרים, לאחר מכן - את הספרות במקום השני מימין, אחר-כך - את הספרות במקום השלישי וכן הלאה. בכל מחזור מחשבים את הסכום של שלושה ערכים: סיפרה אחת מכל אחד משני המחוברים ונשא (Carry). ערך הנשא (הסיפרה להעברה) בכל מחזור שווה לסיפרה השניה מימין של הסכום (בחיבור עשרוני זוהי סיפרת העשרות). אם אין העברה, ערך הנשא הוא '0'.

התוצאה היא מספר המורכב מספרות האחדות של סכום כל מחזור. אם בתום התהליך ערך הנשא שונה מ-'0', נוסף ערך זה משמאל למספר.

נציג תכנית לחיבור שני מספרים בינריים, שתחום הספרות שלהם הוא 0 או 1. אם אורך אחד המחוברים קטן יותר מחברו, מופעלת הפונקציה MILOUI להוספת אפסים משמאל למספר הקצר, על מנת ששניהם יהיו באורך זהה.

הגדרת ADDITION

```
TO ADDITION :A :B
TEST COUNT :A = COUNT :B
IFT [OP SUMBIN :A :B 0 " ]
TEST COUNT :A > COUNT :B
IFT [OP SUMBIN :A (MILOUI :A :B) 0 " ]
OP SUMBIN (MILOUI :B :A) :B 0 " ]
END
```

הגדרת MILOUI

```
TO MILOUI :N1 :N2
IF COUNT :N1 = COUNT :N2 [OP :N2]
OP MILOUI :N1 (WORD "0 :N2)
END
```

הפונקציה ADDITION פונה לפונקציה SUMBIN לחישוב הסכום ומעבירה לה 4 ערכים: שני ערכי המחוברים, '0' עבור הנשא, והמילה הריקה "" כערך התחלתי לתוצאה.

- אם אורך שני המחוברים A ו-B שווה, הם מועברים כפי שהם.
- אם 'COUNT :A < COUNT :B', מועבר ערכו של B בתוספת אפסים מצדו השמאלי, באמצעות הביטוי 'MILOUI :A :B'.
- אחרת, אם אורך של A: הוא הקצר, מועבר ערכו של A: על-ידי חישוב הביטוי 'MILOUI :B :A'.

הגדרת SUMBIN

```
TO SUMBIN :A :B :C :R
TEST EMPTYP :A
IFT [IF :C > 0 [OP WORD :C :R] [OP :R]]
OP SUMBIN BL :A BL :B (CARRY SM) (RESULT SM)
END
```

פונקציה זו מחברת שני מספרים בינריים המוצגים במשתנים A: ו-B:. מעבירים אליה שני מחוברים אשר האורך שלהם זהה ולכן, התנאי לסיום הפונקציה הוא כאשר אין יותר ספרות באחד מהם:
'IF EMPTYP :A'

בפניה אל הפונקציה בכל מחזור משמיטים את הספרה הימנית מכל אחד משני הנתונים, באמצעות ההוראה BL.
אל המשתנה C: מועבר הערך '0' כערך הנשא ההתחלתי, ובכל מחזור חדש הוא מחושב על-ידי הפונקציה **CARRY** הפועלת על התוצאה של **SM**.

בתחילה, ערך התוצאה היא המילה הריקה, והיא מועברת למשתנה R:.
בכל מחזור חדש מחושבת התוצאה החדשה על-ידי הפונקציה **RESULT**

הפועלת על הערך של **SM**. בתום הביצוע, אם אין נשא, התוצאה היא הערך של R:. אחרת - מתוסף ערך הנשא לצדו השמאלי של הערך של R:.

הגדרת SM

```
TO SM
OP (SUM LAST :A LAST :B :C)
END
```

ערך הפונקציה SM הוא סכום הסיפרה הימנית בכל אחד משני המספרים יחד עם הנשא. ערך זה משמש כקלט גם ל-CARRY, וגם ל-RESULT.
שים לב: כאן לא התבצעה העברת פרמטרים. A:, B: ו-C: הם משתנים לא-מקומיים ולכן הם מוכרים על-ידי הליכים ברמה נמוכה יותר מההליך שבו הם מוגדרים.

הגדרת CARRY

```
TO CARRY :S
IF :S > 1 [OP 1] [OP 0]
END
```

סכום של 3 ספרות בינריות יכול להיות 00 (ערך עשרוני = 0), 01 (ערך עשרוני = 1), 10 (ערך עשרוני = 2) או 11 (ערך עשרוני = 3). הסיפרה השניה מימין במספר הבינרי היא הנשא. ערכה הוא 1 כאשר הסכום גדול מ-1, אחרת היא 0.

הגדרת RESULT

```
TO RESULT :S
IF OR :S = 0 :S = 2 [OP WORD "0 :R]
OP WORD "1 :R
END
```

סיפרת האחדות היא 0 אם הסכום הוא 0 או 2. לכן, מוסיפה פונקציה זו '0' לצדה השמאלי של תוצאה R:, אחרת היא מוסיפה '1'.

* * * * *

18. הגדר פונקציה לחיסור של שני מספרים בינריים חיוביים, כשהמחוסר גדול מן המחסר.

דרך אחת לפתרון התרגיל היא על-ידי השימוש בשיטת המשלים ל-2 עבור המספר המחסר, וחישוב הסכום, מבלי להוסיף את הנשא האחרון.

חישוב המשלים ל-2: הפוך כל 0 המופיע במספר ל-1, וכל 1 ל-0. לאחר מכן הוסף 1 למספר המתקבל (שספרותיו הפוכות).

לדוגמה: המספר הנתון הוא 11100110

הפוך את הספרות ל-00011001

חבר 1 +

ותקבל: 00011011

19. הגדר פונקציה לחיבור שני מספרים בכל בסיס שהוא.

הערה: לצורך החישוב יש להסב את הערכים המיוצגים על-ידי אותיות (אם הם נכללים בתחום).

20. הגדר תכנית המחשבת את A: בחזקת X: מבלי להציג זאת בשיטה המדעית.

21. בהנחה שלא מוגדר הפרדיקט NUMBERP, הגדר אותו בעצמך.

* * * * *

פרויקט:

מכונה להדפסת שיקים במספרים ובמילים

ערך מספרי אנו מבטאים בדרך כלל על-ידי יצוגו בסמלי הספרות. אבל בשימושים רבים, כמו בשיקים למשל, מצאו לנכון לרשום ערך זה גם במילים, כדי למנוע אי בהירות ביחס לסכום הנקוב. ניישם זאת

בתכנית, המקבלת מספר טבעי המורכב מ-6 ספרות ומתרגמת אותו למילים. נגדיר תכנית זו לפי כללי התכנות הפונקציונלי.

כפי שידוע, ערך הסיפרה במספר תלוי במיקום שלה. במספר בן 3 ספרות נמצא את סיפרת היחידות, סיפרת העשרות וסיפרת המאות. מהסיפרה הרביעית ועד השישית המספר נחשב באלפים, אבל גם בשלוש הספרות האלו יש אחדות אלפים, עשרות אלפים ומאות אלפים. כלומר, אם נחלק מספר בן שש ספרות לשלשות, התרגום לשני החלקים יהיה זהה בהבדל אחד - לחלק השמאלי אנו מוסיפים את המילה 'אלפים'.

הגדרת DIG.TO.WRD

```
TO DIG.TO.WRD :NUM
IF :NUM = 0 [OP "ZERO]
OP SE (TRGM.ELEF (LST.ELEF :NUM))                              !
                              (TRGM.MEOT (LST.MEOT :NUM))
END
```

פונקציה זו מקבלת מספר ובודקת אם ערכו הוא '0':

- אם כן, הפלט הוא המילה 'ZERO'.
- אחרת, היא מאחדת למשפט אחד את תרגום האלפים עם תרגום המאות.

תרגום האלפים מתבצע על-ידי הפונקציה **TRGM.ELEF**. הקלט שלה הוא רשימת הפלט של הפונקציה **LST.ELEF**, המחשבת את החלק המציין את האלפים במספר, מהסיפרה הרביעית ועד השישית.

תרגום המאות מתבצע על-ידי הפונקציה **TRGM.MEOT**. הקלט שלה הוא רשימת הפלט של הפונקציה **LST.MEOT**, שהיא רשימת הספרות הראשונות, עד הספרה השלישית.

לדוגמה, אם ערכו של NUM: הוא 340017, אז הפלט של LST.ELEF הוא הרשימה [3 4 0], אשר משמשת קלט עבור הפונקציה TRGM.ELEF, לשם תרגום למילים. הפלט של LST.MEOT הוא הרשימה [1 7], אשר משמשת קלט עבור הפונקציה TRGM.MEOT, לשם תרגום למילים.

הבה נראה כיצד מחושבות שתי רשימות הפלט, המשמשות כל אחת קלט עבור פונקציה אחרת.

הגדרת LST.ELEF

```
TO LST.ELEF :MIS
OP PIRUK (REMAINDER (INT :MIS / 1000) 1000)
END
```

הגדרת LST.MEOT

```
TO LST.MEOT :MIS
OP PIRUK (REMAINDER :MIS 1000)
END
```

כפי שרואים, שתי פונקציות אלו פונות לפונקציה **PIRUK**, אשר בונה רשימה שאיבריה הם הספרות המרכיבות את כל אחד משני חלקי המספר.

הגדרת PIRUK

```
TO PIRUK :WRD
IF EMPTYP :WRD [OP []]
OP FPUT FIRST :WRD PIRUK BF :WRD
END
```

לדוגמה: כאשר נרשום ---> `?SHOW LST.ELEF 307015`

הפלט יהיה: `[3 0 7]`

שים לב - אם אין אלפים, הפלט הוא '[0]'.

וכאשר נרשום ---> `?SHOW LST.MEOT 307015`

הפלט יהיה (ללא ה-0 משמאל): `[1 5]`

הפלט של LST.ELEF משמש קלט עבור TRGM.ELEF, ואילו הפלט של LST.MEOT משמש קלט עבור TRGM.MEOT.

הגדרת TRGM.ELEF

```
TO TRGM.ELEF :LST
IF :LST = [0] [OP " ] [OP (SE TRGM.MEOT :LST "THOUSAND)]
END
```

אם אין אלפים, נקבל כפלט את המילה הריקה '"'. אחרת, יש לבצע תרגום זהה לתרגום של המאות, אבל יש לציין במילים שהערך הוא באלפים. במקרה זה נוסיף למשפט הפלט את המילה '"THOUSAND'.

הגדרת TRGM.MEOT

```
TO TRGM.MEOT :LST
IF COUNT :LST < 3 [OP ASAROT :LST]
TEST AND (LAST :LST) = 0 (FIRST BF :LST) = 0
IFT [OP (SE UNIT FIRST :LST "HUNDRED)]
IFF [OP (SE UNIT FIRST :LST "HUNDRED "AND ASAROT BF :LST]
END
```

אם אין סיפרת מאות, התוצאה תהיה הפלט של הפונקציה ASAROT.
אם יש סיפרת מאות, יש לבדוק אם גם סיפרת העשרות וגם סיפרת היחידות הן אפס.

- אם כן, סיפרת המאות תתורגם על-ידי הפונקציה UNIT, ונציין במילים שערכה הוא במאות: '"HUNDRED'.
- אחרת, יש להרכיב משפט המורכב מתרגום סיפרת המאות יחד עם התרגום ש-ASAROT תבצע לשתי הספרות הנותרות.

הגדרת ASAROT

```
TO ASAROT :L
IF COUNT :L = 1 [IF FIRST :L = 0 [OP " ] [OP UNIT FIRST :L]]
IF FIRST :L = 0 [OP UNIT LAST :L]
IF FIRST :L = 1 [OP ESRE (LAST :L)]
TEST LAST :L = 0
IFT [OP TENS (FIRST :L)]
IFF [OP (SE TENS (FIRST :L) UNIT (LAST :L))]
END
```

א. אם הקלט של ASAROT הוא בן סיפרה אחת, אז - אם היא שווה ל-'0', יש להחזיר מילה ריקה, אחרת - הפלט יהיה תרגום של סיפרת יחידות זו על-ידי הפונקציה **UNIT**.

ב. אם הקלט הוא בן שתי ספרות, אז -
- אם סיפרת העשרות היא '0', הפלט יהיה תרגום סיפרת היחידות באמצעות **UNIT**.
- אם סיפרת העשרות היא '1', הפלט יתקבל מביצוע התרגום על-ידי הפונקציה **ESRE**.
- אם סיפרת היחידות היא '0', הפלט יהיה תרגום הפונקציה TENS. אחרת, הפלט יהיה משפט המורכב מתרגום סיפרת העשרות על-ידי **TENS**, יחד עם התרגום של סיפרת היחידות על-ידי UNIT.

נציג יחד את שלוש הפונקציות UNIT, ESRE ו-TENS, בזו אחר זו. לכל אחת מהן נצרף **הערה**, כדי לציין את ייעוד התכנית, או לתת הסברים כלשהם. מציינים הערה בגוף התכנית באמצעות ההליך REM, כפי שמוגדר לפניך:

```
TO REM :REMARK
END
```

ההליך REM מקבל קלט כלשהו, גם רשימה, אך לא מבצע כל הוראה. המחשב מתעלם מהערך של הקלט. דוגמאות להוראת REM בגוף התכנית נראה בהמשך.

הגדרת UNIT

```
TO UNIT :N
REM ["UNIT" IS FOR YEHIDOT]
OP ITEM :N [ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE]
END
```

הגדרת ESRE

```
TO ESRE :N
REM ["ESRE" IS FOR NUMBERS FROM 10 TILL 19]
OP ITEM :N + 1 [TEN ELEVEN TWELVE THIRTEEN FOURTEEN          !
            FIFTEEN  SIXTEEN SEVENTEEN EIGHTEEN NINETEEN]
END
```

הגדרת TENS

```
TO TENS :N
REM ["TENS" IS FOR ASAROT]
OP ITEM :N - 1 [TWENTY THIRTY FORTY FIFTY SIXTY SEVENTY     !
                                           EIGHTY NINETY]
END
```

כאשר נרשום --->

```
?PR DIG.TO.WRD 340017
```

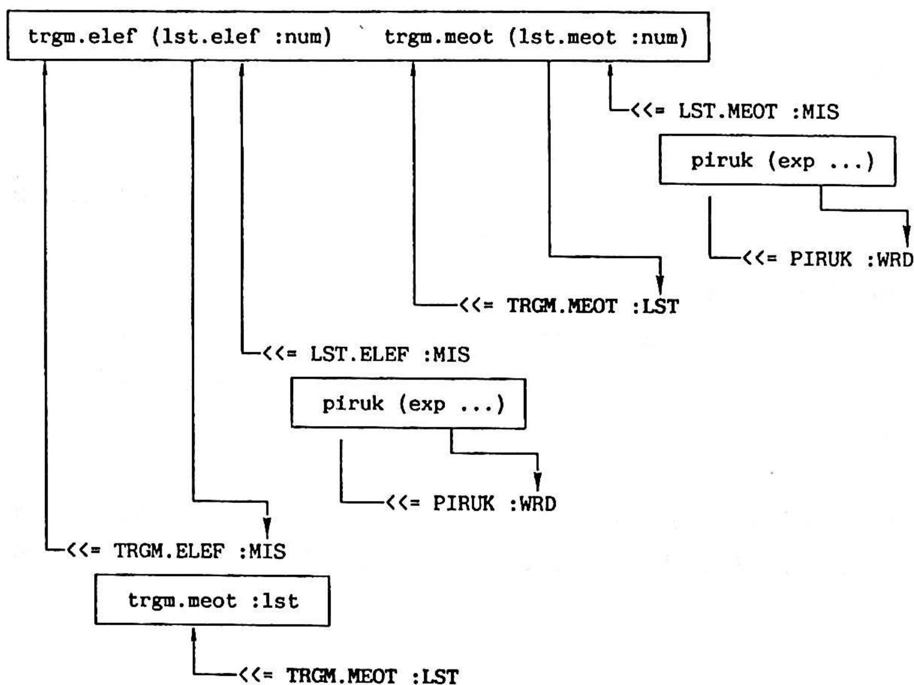
הפלט יהיה:

```
THIRTY HUNDRED AND FORTY THOUSAND SEVENTEEN
```

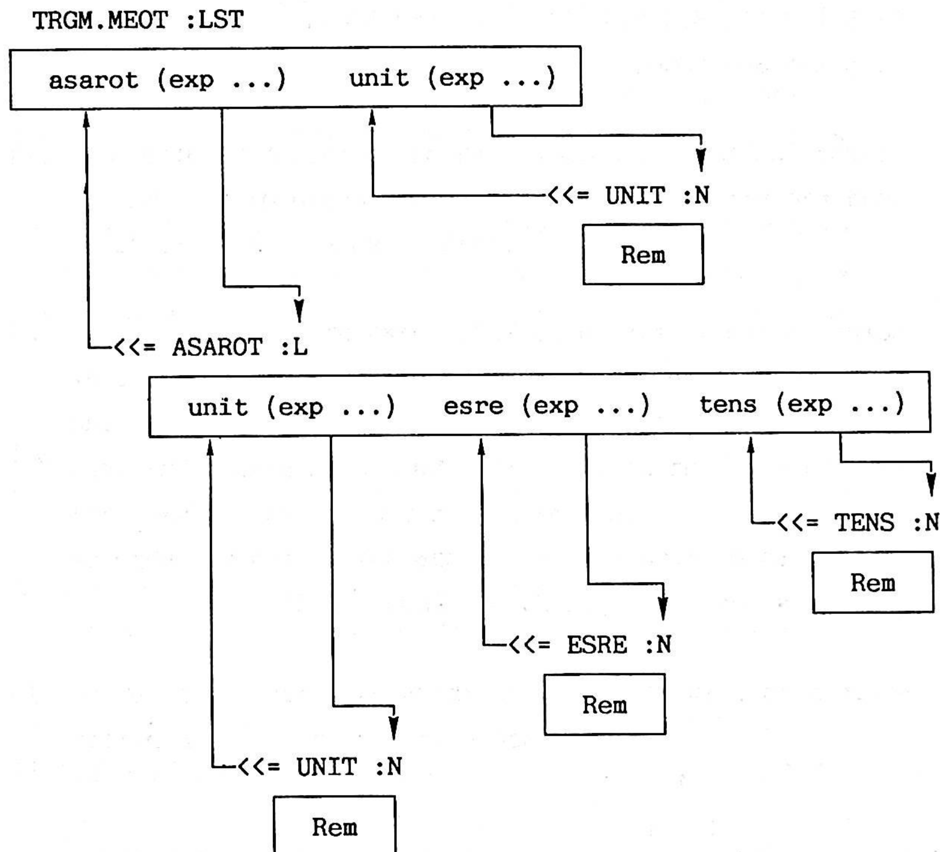
בשני התרשימים הבאים נציג בשני חלקים את שלבי הביצוע של התכנית להדפסת שיקים.

א. התרשים מתאר את החלק הראשון של התכנית, עד הפניה לפונקציה TRGM.MEOT. הפניה ל-TRGM.MEOT מתבצעת **פעמיים**.

DIG.TO.WRD :NUM



ב. התרשים מתאר את מהלך הביצוע של הפונקציה TRGM.MEOT.



* * * * *

תרגילים (המשך)

22. שנה את התכנית DIG.TO.WRD, כדי שתתרגם מספר שבו עד 9 ספרות. בנוסף, שפר אותה, כך שתבצע תחילה בדיקה אם הקלט הוא אכן מספר המורכב רק מספרות, ושמספר הספרות אינו עולה על 9.

23. הגדר פונקציה המקבלת רשימה בת רמות אחדות, ומחזירה את כל האטומים הנמצאים ברמה 1.

לדוגמה,

אם הקלט הוא: [1 5 [ER FD] MON SUN [[ADD[10]]BK] LOGO]

הפלט יהיה: [1 5 MON SUN LOGO]

24. הגדר פונקציה המקבלת רשימה בת רמה כלשהי, ומחזירה ברשימה את כל האטומים הנמצאים ברמה ה-N:. אם רשימת הקלט היא ברמה נמוכה מ-N:, מוחזרת רשימה ריקה.

25. הגדר פונקציה המקבלת רשימה LST: בת n רמות, ומחזירה רשימה שאיבריה הם רשימות בנות רמה אחת. איברי הרשימה האחת יהיו האטומים המופיעים ברמה 1 של LST:, איברי הרשימה השניה יהיו האטומים הנמצאים ברמה 2 של LST:, ... ואיברי הרשימה ה-n יהיו האטומים המופיעים ברמה ה-n של LST:.

אם הקלט הוא כמו בדוגמה שבתרגיל 23, אז הפלט יהיה:

[[1 5 MON SUN LOGO] [ER FD BK] [ADD] [10]]

26. משולש פסקל מציג את מקדמי הבינום של ניוטון. כתוב תכנית לבניית משולש פסקל לפי המבנה הבא:

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5  10  10   5   1
  ...    ...    ...
```

מספר האיברים בכל שורה שווה למספר השורה, האיבר הראשון והאחרון הם '1', ושאר האיברים מתקבלים על-ידי חיבור שני האיברים שמעליהם.

לדוגמה, הערך של האיבר השלישי בשורה 6: הוא 10, כי 10=4+6.

27. הגדר פונקציה המקבלת רשימה ברמה אחת LST:, ומחזירה רשימה במספר רמות כמספר איברי LST:. האיבר הראשון יופיע ברמה 1 של רשימת הפלט, האיבר השני ברמה 2 של רשימת הפלט והאיבר ה-n יופיע ברמה n של רשימת הפלט, לפי הסדר.

לדוגמה, אם הקלט הוא: [ALEF BEIT GIMEL DALET HEI]

אז נקבל: [ALEF [BEIT [GIMEL [DALET [HEI]]]]]

28. הגדר פונקציה כמו בתרגיל 27, אשר מקבלת רשימה בת n איברים. אבל, בפונקציה זו האיבר הראשון יופיע ברמה ה-n של רשימת הפלט, האיבר השני יופיע ברמה 'n - 1' של רשימת הפלט, והאיבר ה-n יופיע ברמה 1 של רשימת הפלט.

לדוגמה, אם הקלט הוא: [ALEF BEIT GIMEL DALET HEI]

נקבל: [[[[[ALEF] BEIT] GIMEL] DALET] HEI]

29. כתוב תכנית המקבלת רשימת מספרים בסדר עולה, ומחזירה איברים אלה ברשימה, אשר כל מספר מופיע בה ברמה התואמת את ערכו.

לדוגמה, אם הקלט הוא: [1 3 3 4 5 5 5 7]

נקבל: [1 [[3 3 [4 [5 5 5 [[7]]]]]]]

30. הגדר תכנית כמו בתרגיל 29, אך הפעם רשימת הקלט תהיה בסדר יורד.

אם הקלט הוא: [7 5 5 5 4 3 3 1]

נקבל: [[[[[[[7]] 5 5 5] 4] 3 3]] 1]

פרק 14

תכניות אינטראקטיביות

אחד היתרונות של שפת תכנות הוא, שניתן לכתוב בה תכנית בעלת אופי שיחתי בין המשתמש לבין המחשב. כלומר, במהלך ביצוע התכנית מתבקש המשתמש להקיש נתון או נתונים, אשר משמשים כקלט לביצוע המשך התכנית.

התכניות יכולות לשמש עבור משחקי אתגר או שעשוע, למשחקים לימודיים, ללומדות מכל הסוגים וגם עבור חישובים עסקיים. בכל מקרה רצוי לכלול בתוך התכנית, באמצעות REM למשל, תיאור קצר והסבר על הפעולות שהיא מבצעת והדרכה על אופן השימוש בכלליה. על התכנית להנחות את המשתמש להזין קלט ולהודיע לו לאיזה סוג של קלט היא מצפה. כאשר התכנית מקבלת נתונים, עליה לבצע בדיקת תקינות של הקלט.

פונקציות קלט של רשימות

את **פונקציית הקלט READCHAR** לקליטת תו כלשהו כבר פגשנו. נכיר כעת פונקציה לקליטת רשימה.

הפונקציה READLIST (RL)

פונקציה זו, כפי שניתן להבין מהשם שלה, הינה הוראת קלט המקבלת נתונים אשר מהווים רשימה. גם כאן, כמו במקרה של הזנת תו בודד, לוגו תפסיק את הביצוע כדי לאפשר למשתמש לספק את הנתונים. בפונקציה RC הביצוע המשיך לאחר הקשה של תו אחד, מפני ש-RC

דורשת תו אחד ויחיד. בפונקציה RL הביצוע ימשיך רק כאשר נקיש על <RET>, על מנת שלוגו תדע שסיימנו להקיש את כל תוכן הרשימה.

לאחר שמזינים את הנתונים, נקבל רשימה אשר משתמשים בה כמו בכל רשימה אחרת. לכן, ההתייחסות לנתונים תעשה על-ידי פונקציות הבחירה, גם אם רשמנו נתון **אחד** ויחיד.

לדוגמה, נוכל לרשום ---> **?PR FIRST RL**

לוגו תמתין, נקיש את הנתונים ובסיום נקיש <RET>.

אם נקיש למשל: 'FATHER MOTHER BROTHER SISTER'

הפלט יהיה: FATHER

הבה נגדיר פונקציה הבונה רשימה מ-NUM: כתובות של תלמידים הנקלטות מן המשתמש באמצעות פונקציה RL. כל כתובת נקלטת כרשימה.

הגדרת ADDRESS

```
TO ADDRESS :NUM
IF :NUM = 0 [OP []]
OP FPUT RL ADDRESS :NUM - 1
END
```

בכל מחזור ימתין המחשב לקליטת נתונים מן המשתמש וימשיך את הביצוע רק לאחר שיוקש <RET>. רשימת הקלט החדשה תוצב בתחילת רשימת הפלט של המחזור הבא. הביצוע יסתיים כאשר יקלטו כל NUM: הכתובות.

* * * * *

תרגילים

1. כתוב הליך המצפה לקלט. בהקשת מקש כלשהו, יודפס קוד ה-ASCII של התו שהוקש.

2. כתוב תכנית המציגה שאלה כלשהי על המסך, ומצפה מן המשתמש שיקיש על אחד המקשים 'Y' (YES) או 'N' (NO).
- אם התו שהוקש הוא 'Y' תודפס המילה YOFFI".
- אם התו הוא 'N', תודפס המילה HAVAL".

הגדר את התכנית, כך שכל עוד לא הוקש אחד משני התווים האלה, תמשיך השאלה להיות מוצגת על המסך, והמחשב ימשיך לצפות לנתון.

3. כתוב תכנית המצפה מן המשתמש לשני נתונים מספריים, ומדפיסה את סכומם.

4. הגדר שוב את התכנית שבתרגיל 3, אך הפעם יש להציג את כל התרגיל החשבוני כשלידו התוצאה.
רמז: מכיון שיש יותר מפעולה אחת המתבצעת על הנתונים הנקלטים, יש לקשור את ערכיהם למשתנים.

5. הגדר תכנית המבקשת קלט מן המשתמש ובודקת אם תוכן רשימת הקלט יכול לשמש משפט פתיחה להגדרת הליך.
כלומר, התכנית בודקת:
- אם רשימת הקלט מתחילה במילה TO,
- האם המילה השניה תקינה כשם של הליך,
- אם יש עוד נתונים, ואם הם מתאימים להיות שמות של משתנים,
אם הקלט יכול לשמש משפט פתיחה להגדרת הליך, על התכנית להדפיס את התו 'כ' בתחילת שורה חדשה.
אם לא, תוצג הודעת שגיאה.

6. הגדר תכנית המשתמשת בפונקציה ADDRESS, אשר תקלוט מן המשתמש NUM: כתובות של תושבים מכל חלקי הארץ, ותדפיס רק את אלה הגרים בעיר CITY:.

7. כתוב תכנית הקולטת רשימת תלמידים וציוניהם במקצוע מסוים. בכל המתנה יקיש המשתמש שלושה נתונים: שם המשפחה, שם פרטי והציון. על התכנית להחזיר רשימה ממויינת בסדר יורד של הציונים, כלומר מן התלמיד בעל הציון הגבוה ביותר עד לתלמיד בעל הציון הנמוך ביותר.

* * * * *

רשימה דו-מימדית

למדנו על רשימה בת רמה אחת (מערך חד-מימדי) וראינו שניתן לטפל ברשימות בכל רמה שהיא, דהיינו, במערך רב-מימדי. נעסוק עתה ברשימה דו-מימדית בת שתי רמות, שהיא מערך דו-מימדי.

מערך דו-מימדי ניתן להצגה בצורת טבלה בת X עמודות ו-Y שורות, אשר נכנה "טבלה בסדר X * Y".

לדוגמה, טבלה בסדר 4*3 (4 עמודות * 3 שורות) ניתנת להצגה כך:

a(1,1)	a(1,2)	a(1,3)	a(1,4)
a(2,1)	a(2,2)	a(2,3)	a(2,4)
a(3,1)	a(3,2)	a(3,3)	a(3,4)

באמצעות טכניקת הרשימות נוכל לתאר את הטבלה כרשימה בת שתי רמות, שיש בה Y רשימות במימד אחד, שבכל אחת מהן X איברים. כך נוכל למשל, להתייחס אל האיבר 'a(i,j)', כאשר i מציין את מספר השורה ו-j מציין את מספר העמודה, באופן הבא:

```
ITEM :J ITEM :I :LIST
```

לדוגמה, הערך של LIST: הוא:

```
[[1 2 3 4][2 4 8 10][3 6 9 12]]
```

הערך של I: הוא 2 והערך של J: הוא 4.

ערך הביטוי יהיה שווה ל-10, שהוא האיבר הרביעי בשורה השניה.

* * * * *

תרגילים (המשך)

8. כתוב תכנית המדפיסה טבלה של מערכת השיעורים של תלמיד. המשתמש מתבקש להזין את המחשב בשמות המקצועות שהוא לומד בכל יום.

לדוגמה: (PR [HADPESS MIKTSOOT YOM] :DAY ":)

אם הערך של DAY: הוא 'RISHON' למשל,

יופיע המשפט הבא על המסך: HADPESS MIKTSOOT YOM RISHON :

המחשב ימתין לקבלת רשימת המקצועות של אותו יום מן המשתמש. פעולה זו תתבצע עבור כל אחד מששת ימי השבוע. כל מקצוע יש לתאר ב-4 תווים.

בתום קליטת כל הנתונים תופיע טבלת המערכת על המסך. לדוגמה,

```
   SUN   MON   TUE   WED   THU   FRI
 ======================================
1  ENGL  BIBL  BIBL  ...   ...   ...
2  MATH  MATH  MATH  ...   ...   ...
3  LOGO  HEBR  LOGO  ...   ...   ...
4  HEBR  TEVA  SPRT  ...   ...   ...
5  HIST        SPRT
 ======================================
```

כלומר, מתחת לשם של כל יום יירשמו המקצועות של אותו יום בעמודה, זה תחת זה.

9. בפסטיכון (פסטיבל בבית-ספר תיכון) אשר נערך במסיבת סיום שנת הלימודים, היה על המארגנים להכריז על השיר אשר זכה בתחרות.

כל אחת מארבע הכיתות (ט', י', י"א ו-י"ב) מסרה את רשימת הנקודות, אשר העניקה לכל אחד מ-4 השירים לפי סדר הופעתם (שיר מספר 1, שיר מספר 2 וכו'). הניקוד נע בין 1 ל-5, ואסור היה לתת לשני שירים מספר נקודות זהה.

כתוב תכנית אשר תקבל רשימה אחר רשימה מן המשתמש ותדפיס טבלה להצגת תוצאות הבחירה, לדוגמה:

	SHIR1	SHIR2	SHIR3	SHIR4
5TH	5	4	2	3
6TH	3	5	1	2
7TH	4	3	2	1
8TH	3	5	2	4

על התכנית לחשב ולהדפיס הודעה בדבר השיר הזוכה.

* * * * *

פרויקט:

תכנית ל"ניחוש" הקלף שנבחר

נגדיר כעת תכנית "המנחשת" קלף שנבחר על-ידי השחקן מתוך 21 קלפים כלשהם.

התכנית חוזרת על הפעולות הבאות 3 פעמים:

- התכנית מסדרת את הקלפים ב-3 שורות, דהיינו 7 קלפים בכל שורה.
- השחקן מקיש את מספר השורה בה נמצא הקלף הנבחר.
- התכנית "טורפת" את הקלפים.

לאחר מכן, היא מדפיסה את הקלף שהשחקן בחר. קלף זה יימצא באמצע השורה השניה בתום הביצוע.

ערבוב הקלפים נעשה לפי הכללים הבאים:

א. יש לאסוף את הקלפים שורה אחר שורה, ולהקפיד שהשורה בה הקלף נמצא תאסף שניה בתור.

ב. את חבילת הקלפים שנאספה יש לסדר עמודה אחר עמודה, כלומר מלמעלה למטה, שלושה קלפים בכל עמודה.

התכנית GUESSCARD מבצעת את ניחוש הקלף הנבחר, באמצעות פונקציות עזר שונות, כפי שנראה להלן.

הגדרת GUESSCARD

```
TO GUESSCARD
PR ITEM 11 CRD CRD CRD (CHOOSE [])
END
```

הקלט של **CRD** הינו רשימה במימד אחד של 21 הקלפים. בפעם הראשונה זוהי רשימה שאיבריה נבחרו באופן אקראי מתוך 52 קלפים על-ידי הפונקציה **CHOOSE**. הפלט של CRD הינו רשימה במימד אחד, של הקלפים לאחר ערבוב. פלט זה משמש קלט עבור CRD לערבוב שני, ולאחר מכן לערבוב שלישי. מכיון שהפלט הסופי מוצג למחשב כרשימה במימד אחד, נמצא הקלף הנבחר במקום ה-11 ברשימה.

בחירת הקלפים באופן אקראי מתבצעת על-ידי תכנית המורכבת משלוש פונקציות: CHOOSE, CHS ו-CARD52.

הפונקציה **CARD52** כוללת את רשימת 52 הקלפים, שמתוכם בוחרים את 21 הקלפים שישתתפו במשחק.

הגדרת CARD52

```
TO CARD52
REM [C=CLUB H=HEART S=SPADE D=DIAMOND]
REM [0=10 J=JACK Q=QUEEN K=KING]
OP [1C 2C 3C 4C 5C 6C 7C 8C 9C 0C JC QC KC        !
    1H 2H 3H 4H 5H 6H 7H 8H 9H 0H JH QH KH        !
    1S 2S 3S 4S 5S 6S 7S 8S 9S 0S JS QS KS        !
    1D 2D 3D 4D 5D 6D 7D 8D 9D 0D JD QD KD]
END
```

השתמשנו בהליך **REM**, על מנת להסביר את משמעות הסמלים.

הגדרת CHOOSE והגדרת CHS

```
TO CHOOSE :CD21
IF COUNT :CD21 = 21 [OP :CD21]
OP CHOOSE (CHS (ITEM 1 + RANDOM 52 CARD52))
END
```

```
TO CHS :C
IF MEMBERP :C :CD21 [OP :CD21]
OP FPUT :C :CD21
END
```

בתחילה, הקלט של CHOOSE הוא הרשימה הריקה (ראה ב-GUESSCARD). כל עוד אין ברשימת הקלט CD21: 21 קלפים, פונה CHOOSE ל-CHS. הקלט עבור C: הוא ערך שנבחר באופן אקראי מתוך CARD52.

CHS בודקת אם ערך זה נבחר כבר קודם. אם כן - היא מחזירה את הרשימה CD21: כפי שהיא. אחרת, היא מציבה ערך זה ברשימה.

הביצוע מסתיים כאשר נמצאים ברשימה 21 קלפים, ואז הפונקציה CHOOSE מחזירה רשימה זו כפלט.

פלט זה, הערוך ברשימה במימד אחד, משמש קלט עבור הפונקציה CRD בפעם הראשונה.

הגדרת CRD

```
TO CRD :L21
OP PRNTCARD (KLAFIM :L21 7 [])
END
```

הפלט של הפונקציה CRD מחושב על-ידי הפונקציה **PRNTCARD**, הפועלת על קלט שהוא רשימת הפלט של הפונקציה **KLAFIM** אשר נציג בהמשך. הפלט של CRD ישמש כקלט עבור ההפעלה הבאה של הפונקציה CRD.

KLAFIM היא פונקציה המקבלת רשימה LST: במימד אחד, והופכת אותה לרשימה דו-מימדית. כלומר, היא הופכת רשימה אחת לרשימה של רשימות, שבכל אחת מהן NUM: איברים (במילים אחרות - טבלה בת NUM: עמודות).

הגדרת KLAFIM

```
TO KLAFIM :LST :NUM :L
IF EMPTYP :LST [OP LIST :L]
TEST COUNT :L = :NUM
IFT [OP FPUT :L KLAFIM :LST :NUM []]
OP KLAFIM BF :LST :NUM (LPUT (FIRST :LST) :L)
END
```

המשתנה L: משמש כמחסנית, אשר בתחילת הביצוע היא ריקה. בכל מחזור מאחסנים בה איבר נוסף מהרשימה LST:

```
(LPUT (FIRST :LST) :L)
```

עד שמספר האיברים מגיע ל-NUM: -

```
TEST COUNT :L = :NUM
```

עתה מציבים מחסנית זו, שהיא רשימה בת NUM: איברים, בתחילת רשימת הפלט:

```
OP FPUT :L KLAFIM :LST :NUM []
```

הביצוע מסתיים כאשר אין יותר איברים ב-LST:, ואז משמשת המחסנית של המחזור האחרון כנקודת מוצא לחישוב הפונקציה:

```
IF EMPTYP :LST [OP LIST :L]
```

אם לדוגמה נרשום --->

```
?PR KLAFIM [A B C D E F] 3 []
```

נקבל את הפלט הבא:

```
[[A B C] [D E F]]
```

הפונקציה PRNTCARD פועלת על הפלט המוחזר מ-KLAFIM, אשר מהווה קלט עבור המשתנה L3X7:.

הגדרת PRNTCARD

```
TO PRNTCARD :L3X7
CLEARTEXT
SETCURSOR [4 3] TYPE FIRST :L3X7
(PR "\ \ \=\=\> [LINE NO 1])
SETCURSOR [7 3] TYPE ITEM 2 :L3X7
(PR "\ \ \=\=\> [LINE NO 2])
SETCURSOR [10 3] TYPE LAST :L3X7
(PR "\ \ \=\=\> [LINE NO 3])
REPEAT 10 [PR "\ ]
PR [\>\>\> PRESS THE NO OF THE LINE WHERE \<\<\< ]
REPEAT 10 [TYPE " ]
PR [\>\> YOUR CARD IS \<\< ]
OP POINT RC :L3X7
END
```

פונקציה זו עורכת את המסך ומדפיסה את הטבלה שורה אחר שורה. היא מנחה את המשתמש להקיש את המספר של השורה שבה נמצא המספר הנבחר. לאחר מכן היא פונה לפונקציה POINT, אשר ממתינה לקבלת הקלט מן המשתמש, באמצעות RC (READCHAR). **זכור**, אם אתה עובד ב-APPLE LOGO II, או ב-APPLE LOGO, יש להפוך את סדר הקואורדינטות.

הפונקציה POINT מחשבת את הפלט עבור הפונקציה PRNTCARD, הפלט הזה הוא למעשה ערבוב הקלפים בהתאם לתשובת השחקן.

הגדרת POINT

```
TO POINT :P :R
TEST MEMBERP :P [1 2 3]
IFF [OP POINT RC :R]
IF :P = 1 [OP MAT3 (SE ITEM 2 :R FIRST :R LAST :R)]
IF :P = 2 [OP MAT3 (SE FIRST :R ITEM 2 :R LAST :R)]
OP MAT3 (SE FIRST :R LAST :R ITEM 2 :R)
END
```

המשתנה :P מקבל את ערך הנתון של המקש שנלחץ. POINT תמשיך בביצוע רק אם ערך הקלט הוא '1' או '2' או '3' - TEST MEMBERP :P [1 2 3]

אחרת, היא תמתין למקש אחר - [IFF [OP POINT RC :R]

המשך הביצוע הוא איסוף הקלפים לרשימה במימד אחד. סדר האיסוף תלוי בערכו של :P. השורה ה-:P-י תמיד תיאסף שניה בתור. רשימה זו תהווה קלט למשתנה :VEC של הפונקציה **MAT3**.

הגדרת MAT3

```
TO MAT3 :VEC
OP KLAF (KLAFIM :VEC 3 []) 1
END
```

MAT3 פונה לפונקציה **KLAF**, להמשך ערבוב הקלפים. הפעולות האלו מכוונות לסדר את הרשימה בטבלה באופן שבו היא היתה נוצרת, אילו סידרנו את הרשימה עמודה אחר עמודה.

לצורך זה, KLAF מקבלת רשימה של שבע רשימות. בכל רשימה שלושה איברים (:M7X3), אשר הוכנו על-ידי הפונקציה **KLAFIM**, שהגדרנו לעיל.

הגדרת KLAF

```
TO KLAF :M7X3 :NUM
IF :NUM > 3 [OP []]
OP SE (KL :M7X3 :NUM) (KLAF :M7X3 :NUM + 1)
END
```

KLAF מאחדת בכל מחזור את הפלט של **KL**, עם הפלט של המחזור הבא לרשימה אחת. היא עושה זאת 3 פעמים.

הגדרת KL

```
TO KL :MX :INDEX
IF EMPTYP :MX [OP []]
OP FPUT (ITEM :INDEX FIRST :MX) (KL BF :MX :INDEX)
END
```

הפלט של KL הינו רשימה, שאיבריה הם כל האיברים המופיעים במקום ה-INDEX:, בכל אחת משבע הרשימות.

הרצת התכנית:

להרצת התכנית עלינו לרשום ---> ?GUESSCARD

* * * * *

בתכנית GUESSCARD השתמשנו ברשימה דו-מימדית. כדי לבצע את המשימה של ניחוש הקלף אפשר להשתמש ברשימה חד-מימדית, כפי שנציג בתכנית הבאה GUESSCARD2. אך לפני כן, הבה נכיר את הפונקציה CURSOR.

הפונקציה CURSOR

פונקציה זו מחזירה רשימה ובה שני ערכי הקואורדינטות [x y], המורים על מקומו של הסמן. האיבר Y מציין את מספר השורה, והאיבר X - את העמודה. פונקציה זו, היא דוגמה לפונקציה **לא-קבועה**, שאינה פועלת על משתנים.

בפונקציה נוכל להשתמש גם בביטויים חשבוניים. לדוגמה:

```
?CLEARTEXT REPEAT 4 [SETCURSOR SE (3 + FIRST CURSOR)          !
                    (5 + LAST CURSOR) TYPE "UH_HOO ]
```

זכור, ב-APPLE LOGO II וב-APPLE LOGO הערך הראשון מציין את העמודה (הקואורדינטה x), והערך השני (הקואורדינטה y) מציין את השורה.

* * * * *

נדגים דרך אחרת אפשרית להגדיר את GUESSCARD ואת CRD. נכנה אותן GUESSCARD1 ו-CRD1 בהתאמה. השווה את שתי ההגדרות וקבע מהי הדרך הנבחרת על-ידך.

הגדרת GUESSCARD1

```
TO GUESSCARD1
PR ITEM 11 CRD1 (CHOOSE []) 3
END
```

הגדרת CRD1

```
TO CRD1 :L21 :C
IF :C < 1 [OP :L21]
OP CRD1 (PRNTCARD (KLAFIM :L21 7 [])) :C - 1
END
```

כפי שרואים, CRD1 היא פונקציה רקורסיבית הקוראת לעצמה. לכן מתבצעת רק פניה אחת ל-CRD1 מתוך GUESSCARD1.
שים לב לערך שהמשתנה L21: מקבל בכל מחזור.

גם את הפונקציה לבחירת 21 הקלפים, נוכל להגדיר באופן שונה. הנה דוגמה:

הגדרת CHOOSE1

```
TO CHOOSE1
OP BHAR CARD52 21 (1 + RANDOM 52)
END
```

CHOOSE1 מעבירה את ערך הפונקציה **BHAR** לפונקציה CRD1.
שים לב, כי ל-CHOOSE1 אין צורך בקלט.

הגדרת BHAR

```
TO BHAR :L :N :R
IF :N < 1 [OP []]
OP FPUT ITEM :R :L                                                    !
  BHAR (DELETE :R :L) :N - 1 (1 + RANDOM (COUNT :L) - 1)
END
```

בקריאה ל-BHAR מועברת רשימת 52 הקלפים למשתנה L:, ל-N: מועבר המספר 21, ול-R: מועבר מספר אקראי בתחום (1..52). בכל מחזור מוצב האיבר ה-R: של L: בתחילת רשימת הפלט של המחזור הבא.
בכל מחזור חדש מקבל המשתנה L: את רשימת הקלפים ללא האיבר ה-R: של המחזור הנוכחי. הסרת איבר זה מן הרשימה מתבצעת על-ידי הפונקציה DELETE, כדי שלא ייבחר קלף כלשהו פעמיים. ערך המשתנה N: מופחת באחד, ו-R: של המחזור הבא מקבל מספר אקראי חדש בתחום ("מספר הקלפים הנותרים"..1).

הגדרת DELETE

```
TO DELETE :R :LL
IF :R = 1 [OP BF :LL]
OP FPUT FIRST :LL (DELETE :R - 1 BF :LL)
END
```

המשתנה R: משמש כמונה בשיטת הספירה לאחור. כל עוד אין ערכו שווה ל-'1' מוצב האיבר הראשון של הרשימה בתחילת רשימת הפלט.
במחזור הראשון, כאשר R: הוא '1', מוחזרת רשימת האיברים הנותרים ללא האיבר הראשון, שהוא האיבר ה-R:. זהו הערך ההתחלתי לחישוב כולו.

* * * * *

פרויקט:

ניחוש קלף - רשימה חד-מימדית

את התכנית לניחוש קלף אפשר להגדיר גם באמצעות רשימה חד-מימדית. עלינו יהיה להחליף כמה מהפונקציות שהגדרנו בפרויקט הקודם, כאשר עסקנו ברשימות דו-מימדיות.

הגדרת GUESSCARD2

```
TO GUESSCARD2
PR ITEM 11 CRD2 CRD2 CRD2 (CHOOSE [])
END
```

פונקציה זו דומה ל-GUESSCARD. היא פונה ל-CRD2 במקום ל-CRD, אך הפונקציות הקודמות CARD52, CHOOSE ו-CHS נשארות ללא שינוי.

הגדרת CRD2

```
TO CRD2 :L
CT SETCURSOR [4 3]
PRKLF :L 7 1
REPEAT 7 [PR " ]
PR [\>\>\> PRESS THE NO OF THE LINE WHERE \<\<\< ]
REPEAT 10 [TYPE "\ ]
PR [\>\> YOUR CARD IS \<\< ]
OP POINT2 RC :L
END
```

CRD2 מציגה באמצעות PRKLF את 21 הקלפים על המסך ומאפשרת להקיש את מספר השורה בה נמצא הקלף הנבחר. אחר-כך היא פונה ל-POINT2 לערבוב הקלפים, בהתאם לתשובת המשתמש.

הגדרת PRKLF

```
TO PRKLF :L3X7 :X :N
IF EMPTYP :L3X7 [(PR "\=\=\> [LINE NO] :N) STOP]
IF :X < 1 [HAZEG PRKLF :L3X7 7 :N + 1 STOP]
(TYPE FIRST :L3X7 "\ )
PRKLF BF :L3X7 :X - 1 :N
END
```

הפונקציה PRKLF מקבלת למשתנה L3X7: רשימה של 21 איברים ברמה אחת. באמצעות המשתנה-מונה X: (בשיטת הספירה לאחור), היא מציגה 7 איברים בשורה.

בתום כל שורה, כאשר X: קטן מ-1, היא פונה ל-**HAZEG** לציון מספר השורה N: על המסך, והצבת הסמן במקום הרצוי להדפסה הבאה.
לאחר מכן מתבצעת קריאה רקורסיבית כשהערך 7 מועבר ל-X:, N: גדל באחד ו-L3X7: נשאר ללא שינוי. שים לב, ההוראה STOP המופיעה בסוף משפט תנאי זה, מסיימת כאן את אותו המחזור.

כל עוד X: גדול מאפס, מוצג האיבר הראשון של הרשימה, והביצוע עובר למחזור הבא. המשתנה L3X7: מקבל 'BF :L3X7', ו-X: קטן ב-1.
הביצוע מסתיים כאשר הרשימה L3X7: הופכת לריקה.

הגדרת HAZEG

```
TO HAZEG
(TYPE "\=\=\> [LINE NO] :N)
SETCURSOR SE (FIRST CURSOR) + 3 3
END
```

הגדרת POINT2

```
TO POINT2 :P :RESH
TEST MEMBERP :P [1 2 3]
IFF [OP POINT2 RC :RESH]
IF :P = 1 [OP SHUROT (SCND :RESH 14)]
IF :P = 2 [OP SHUROT :RESH]
OP SHUROT (SCND :RESH 7)
END
```

הפונקציה POINT2 מקבלת את ערך המקש שהוקש במשתנה P:, ואת רשימת 21 הקלפים - ב-RESH:. השורה שבה נמצא הקלף הנבחר תיאסף שניה ולכן, אם P: הוא 2, הפונקציה POINT2 פונה לפונקציה SHUROT ומעבירה את הרשימה RESH: כפי שהיא.

נוכל לנצל את העובדה שהרשימה היא בת רמה אחת ולהשתמש בפעולת הזזה סיבובית שמאלה. הזזה סיבובית שמאלה פעם **אחת** פירושה: להעביר את האיבר שבמקום n אל מקום n-1, וכך לכל אחד מהאיברים. את האיבר הראשון יש להעביר למקום האחרון ברשימה.

אם P: הוא 1, בפעולת הזזה סיבובית שמאלה 14 פעמים נציב את 7 האיברים הראשונים החל מהמקום השמיני בהתאמה, האיבר הראשון במקום השמיני, האיבר השני במקום התשיעי וכן הלאה. אם הערך של P: הוא 3, בפעולת הזזה סיבובית 7 פעמים נציב את 7 האיברים האחרונים מהמקום השמיני בהתאמה: האיבר ה-15 במקום השמיני, האיבר ה-16 במקום התשיעי וכן הלאה.

פעולת ההזזה מתבצעת על-ידי הפונקציה **SCND**.

- אם P: הוא 1, מועבר ערך הפונקציה SCND כקלט ל-SHUROT לביצוע 14 הזזות - OP SHUROT (SCND :RESH 14)

- אם P: הוא 3, מועבר ערך הפונקציה SCND כקלט ל-SHUROT לביצוע 7 הזזות - OP SHUROT (SCND :RESH 7)

הגדרת SCND

```
TO SCND :L :I
IF :I < 1 [OP :L]
OP SCND (LPUT FIRST :L BF :L) :I - 1
END
```

הפונקציה SHUROT מחזירה את רשימת הקלט במבנה מתאים, לשם הדפסה על-ידי PRKLF. האיברים יופיעו ב-7 עמודות, כשהם מסודרים עמודה אחר עמודה מלמעלה למטה.

בשורה הראשונה יודפסו 7 נתונים, שהראשון בהם הוא האיבר הראשון, לידו האיבר הרביעי, אחר כך השביעי כלומר, בדילוג של 3 איברים מזה שהודפס אחרון.

השורה השניה תתחיל באיבר השני ואחריו יבואו שאר האיברים בדילוג של 3. בשורה השלישית יופיע במקום הראשון האיבר השלישי ואחריו שאר האיברים בדילוג של 3.

הגדרת SHUROT

```
TO SHUROT :L
OP (SE TROF :L 1 TROF BF :L 1 TROF BF BF :L 1)
END
```

SHUROT משתמשת בפונקציה TROF המחשבת את האיברים של כל שורה.

הגדרת TROF

```
TO TROF :L :X
IF EMPTYP :L [OP []]
IF :X = 1 [OP FPUT FIRST :L TROF BF :L 3]
OP TROF BF :L :X - 1
END
```

להרצת התכנית נרשום הפעם ---> ?GUESSCARD2

* * * * *

10. כתוב הליך SETXCOR אשר מקבל קלט למשתנה :X, לקביעת הסמן בעמודה :X, באותה השורה בה הסמן נמצא.

11. כתוב הליך SETYCOR המקבל קלט למשתנה :Y, לקביעת הסמן בשורה :Y, באותה העמודה בה הסמן נמצא.

12. הגדר תכנית המקבלת טבלה, שבה :NUM אי-זוגי של שורות ועמודות, ובודקת אם היא מהווה ריבוע קסם. בריבוע קסם סכום האיברים בכל שורה שווה לסכום האיברים בכל עמודה ולסכום האיברים בכל אלכסון.

13. הגדר תכנית המחזירה פלט של טופס אחד של המשחק BINGO, לפי התיאור הבא:
בשורה הראשונה תופיע המילה BINGO, עם רווחים שווים בין אות לאות. מתחת לזה תופיע טבלה של 5*5, שכל איבר בה נבחר באופן אקראי בתחום (1..75). העמודות תסומנה באותיות השם BINGO, ובכל אחת מהן יהיו מספרים אקראיים מן התחום המותר, כמפורט להלן:

מספר עמודה	כותרת העמודה	מספרים אקראיים בתחום
1	B	1..15
2	I	16..30
3	N	31..45
4	G	46..60
5	O	61..75

אסור שאיבר אחד יופיע פעמיים באותו טופס.

14. כתוב תכנית ליצירת דף של לוח שנה של חודש מסוים. התכנית תחשב כמה ימים יש באותו חודש, ובאיזה יום בשבוע הוא מתחיל. תוכל להיעזר ב-"לוח התאריך" המופיע ב-"יומן לתלמיד". יש המכנים אותו בשם "הלוח התמידי".

15. הגדר תכנית ליצירת ריבוע קסם, עם ערך MIS: אי-זוגי של שורות ועמודות.

אם אין לך רעיון או דרך אחרת, תוכל להיעזר בהנחיות הבאות:

א. הצב את המספר '1' בעמודה האמצעית X: של השורה האחרונה Y:.

ב. אם זה עתה הצבת ערך בשורה האחרונה כאשר MIS: = Y:, עבור לעמודה X+1: של השורה הראשונה (Y=1:) להצבת המספר העוקב הבא.

ג. לאחר הצבת ערך במקום כלשהו יש לעבור לעמודה הבאה (X+1:) של השורה הבאה (Y+1:).

ד. אם זה עתה הצבת ערך בעמודה האחרונה (X=:MIS:), עבור לעמודה הראשונה (X=1:) של השורה הבאה (Y+1:), להצבת המספר העוקב הבא.

ה. אם נמצא כבר ערך במקום הבא בתור, יש לעבור לעמודה X: של השורה Y-1:, ורק שם להציב את הערך הנוכחי.

ו. אם זה עתה הצבת ערך בעמודה האחרונה של השורה האחרונה, כאשר X=:MIS: וגם Y=:MIS:, עבור לעמודה X: שבשורה Y-1: להצבת המספר העוקב הבא.

ז. תהליך זה יסתיים כאשר הערך המוצב יהיה שווה ל-MIS: * MIS:.

* * * * *

הפרדיקט KEYP

פונקציות הקלט READCHAR ו-READLIST מיועדות לקלט של תו או רשימה. בלוגו קיים הפרדיקט **KEYP** אשר מחזיר ערך 'אמת' במקרה של הקשת תו במקש כלשהו.

נניח שבמהלך תכנית מסוימת יש למדוד את משך התגובה של המשתמש. הספירה צריכה להמשך כל עוד לא הוקש תו, ולהסתיים ברגע שהוקש. נכתוב פונקציה אשר תבצע מדידה זו.

הגדרת SFIRA

```
TO SFIRA :N
IF KEYP [PR :N STOP]
SFIRA :N+1
END
```

נרשום ---> ?SFIRA 1

לאחר שהות מה נקיש את האות 'A' למשל, ואז יודפס ערכו של המשתנה N: וגם האות 'A'.

השימוש ב-KEYP לא גורם לתכנית לעצור על מנת לקבל את הקלט הדרוש כמו ב-READLIST, או ב-READCHAR, אלא הביצוע ממשיך עד להקשה של תו כלשהו. ערכו של התו שהוקש נשמר על-ידי KEYP באוגר זמני אשר משמש להעברת נתונים. על ערך זה לא מתבצעת כל פעולה שהיא והוא מועבר אל המסך עם סיום התכנית. כל עוד מכיל האוגר תוכן כלשהו, ערכו של KEYP הוא 'TRUE'.

תנאי הסיום לביצוע ההליך SFIRA הוא 'IF KEYP', אך במשפט זה מותנית גם הדפסת ערכו של N:. אם נשמיט את מילת הסיום 'STOP', הדבר יביא להדפסת כל הערכים של N: המתקבלים במהלך ביצוע ההליך, מרגע ההקשה. מסקנה: ערכו של KEYP הוא TRUE ללא שינוי, כי ערך המקש עדיין שמור באוגר.

ניתן לשחרר את האוגר מן הנתון המוחזק בו על-ידי קליטת הנתון באמצעות הפונקציה RC.

הגדרת SFIRA1

```
?TO SFIRA1 :N
IF KEYP [PR :N RDCH RC STOP]
SFIRA1 :N+1
END
```

הגדרת RDCH

```
TO RDCH :VAR
END
```

כאשר נריץ הפעם את SFIRA1 יודפס רק ערכו של N:, בלי ערך התו שהוקש. ההליך SFIRA1 פונה ל-RDCH כשהוא מעביר למשתנה VAR: את ערכו של המקש באמצעות RC. כך משתחרר האוגר מן התוכן שהיה בו, וערכו של KEYP הופך שוב ל-'FALSE'.

מתעוררת בעיה כאשר המשתמש הוא ילד קטן למשל, אשר לוחץ על מקש ואינו מרפה ממנו מיד. במקרה זה נאספים באוגר תווים אחדים. ההליך הבא מיועד לטפל בשחרור של האוגר מכל תו שהוא.

הגדרת RDCHS

```
TO RDCHS :VAR
IF KEYP [RDCHS RC]
END
```

* * * * *

פרויקט:

דוגמה ליישום לומדה לגיל הרך

התכנית מציגה שלוש אותיות הנבחרות באופן אקראי מרשימת אותיות נתונה. אחת משלוש האותיות שנבחרו משורטטת בשיטת הגרפיקה למחצה בחלקו העליון של המסך. הסמן מדלג עם שהות מה, בין שלוש האותיות שנבחרו. על הילד להקיש מקש כלשהו כאשר הסמן מצביע על האות הזהה לאות המופיעה בראש המסך. כשהתשובה נכונה משורטט SMILY בצדו השמאלי של המסך, אחרת - מופיע CRYLY. התכנית מאפשרת לשחקן לציין את מספר הפעמים שהוא מעונין לשחק, ואת זמן ההשהיה בין פעולות הדילוג של הסמן.

הגדרת PLAY

```
TO PLAY :NUM :TIME
IF :NUM < 1 [STCR 23 1 STOP]
SELECT [] 3 [ALEF BEIT GIMEL ... ... MEM]
RDCHS RC
CT
PLAY :NUM - 1 :TIME
END
```

PLAY ·פונה ל-SELECT לבחירת 3 אותיות מתוך רשימה נתונה, ומשם - להמשך התכנית. בתום כל משחק פונה PLAY ל-RDCHS על מנת להחזיר את הערך 'FALSE' ל-KEYP, כדי שיהיה מוכן לפעם הבאה. הביצוע נפסק כאשר מסיימים את כל NUM: המשחקים.

ההליך STCR הוגדר כקיצור להוראה SETCURSOR:

```
TO STCR :Y :X
SETCURSOR SE :Y :X
END
```

SELECT משתמשת ב-SLT בביצוע רקורסיה עקיפה לבחירת האותיות באופן אקראי. הפניה ל-RMV נעשית כדי למחוק את האות שנבחרה מן הרשימה, ולהבטיח בחירת אות אחרת בכל פעם.

הגדרת SELECT

```
TO SELECT :LST :N :L
IF :N<1 [DRW :LST MAIN :LST (ITEM (RANDOM 3) + 1 :LST) STOP]
SLT (ITEM (RANDOM COUNT :L) + 1 :L) :LST :N :L
END
```

הגדרת SLT

```
TO SLT :OBJ :LIST :N :L
SELECT (FPUT :OBJ :LIST) :N - 1 (RMV :OBJ :L)
END
```

הגדרת RMV

```
TO RMV :OBJ :LL
IF EMPTYP :LL [OP :LL]
TEST :OBJ = FIRST :LL
IFT [OP RMV :OBJ BF :LL]
IFF [OP FPUT FIRST :LL (RMV :OBJ BF :LL)]
END
```

לאחר בחירת שלוש אותיות שונות, פונה SELECT ל-**DRW** לשרטוט האותיות שנבחרו, ולאחר מכן היא פונה ל-**MAIN** (אשר נגדיר בהמשך) להצגת אחת מאותיות אלו בחלק העליון של המסך.

הגדרת DRW

```
TO DRW :LST3
STCR 12 3
REPEAT 1 LIST FIRST :LST3
STCR 12 16
REPEAT 1 LIST FIRST BF :LST3
STCR 12 29
REPEAT 1 LIST LAST :LST3
END
```

בהגדרת DRW השתמשנו **בהוראה REPEAT** כדי לבצע **מספר** פעמים **רשימה** של הוראות. ההוראות ברשימה יכולות להיות שמות של הליכים המוגדרים במחשב.

אם למשל, קיים הליך DOLLARUP -

נוכל לרשום ---> ?REPEAT 1 [DOLLARUP]

מאחר ש-REPEAT פועלת **רק** על רשימה,

נוכל לרשום זאת גם כך - ?REPEAT 1 **LIST** "DOLLARUP

כזכור, הפונקציה LIST בונה רשימה המכילה את ערך הקלט שהיא מקבלת. אם הקלט הוא נתון קבוע, מוחזר נתון זה ברשימה. אם הקלט הוא משתנה, מוחזר **ערכו** של המשתנה ברשימה.

אם לדוגמה, ערך המשתנה X: הוא '777' ערך הביטוי 'X: LIST' יהיה שווה ל-'[777]'. כך הדבר, אם ערכן של המשתנה PROC: הוא המילה 'DOLLARUP' - ערך הביטוי 'PROC: LIST' יהיה שווה '[DOLLARUP]'. במקום הפונקציה LIST אפשר להשתמש בפונקציה SE.

תכונות אלו מאפשרות להגדיר **משתנה**, שערכו הוא לא רק נתון שיש לבצע עליו חישוב. כאשר מציגים משתנה בתוך רשימה, הוא יכול להיות בעצמו חישוב הניתן לביצוע (אלגוריתם).

נניח שלא ננצל את התכונה של REPEAT להפעיל הוראות מתוך רשימה. במקום זאת נפנה להליך BORER, כדי שיבדוק מתוך רשימת כל ה"אותיות" המוגדרות, איזו אות עליו לשרטט.

למשל, במקום לרשום:

```
REPEAT 1 LIST FIRST :LST3
```

נרשום את המשפט הבא:

```
BORER FIRST :LST3
```

את BORER נגדיר כך:

```
TO BORER :ZURA
IF :ZURA = "ALEF [ALEF]
IF :ZURA = "BEIT [BEIT]
IF :ZURA = "GIMEL [GIMEL]
. . .
IF :ZURA = "MEM [MEM]
END
```

בהליך BORER עורכים השוואות כמספר הצורות המוגדרות בתכנית. כלומר, אם מספרן הוא 22, צריכים לרשום 22 משפטי תנאי.

באמצעות REPEAT התכנית הפכה פשוטה יותר, וחסכנו בבדיקות מיותרות. נשתמש בהוראה זו גם בהגדרת התכנית MAIN.
MAIN משרטטת את האות שאתה נעשית ההשוואה, ואחר-כך היא פונה ל-PLY.

הגדרת MAIN

```
TO MAIN :L3 :OBJ
STCR 1 16
REPEAT 1 LIST :OBJ
PLY :L3 :OBJ
END
```

הגדרת PLY

```
TO PLY :L3 :OBJ
STCR 22 6 WAIT :TIME
IF KEYP [BDK (LAST CURSOR) :L3 :OBJ STOP]
STCR 22 19 WAIT :TIME
IF KEYP [BDK (LAST CURSOR) :L3 :OBJ STOP]
STCR 22 32 WAIT :TIME
IF KEYP [BDK (LAST CURSOR) :L3 :OBJ STOP]
PLY :L3 :OBJ
END
```

ב-PLY מדלג הסמן מאות אחת לחברתה, כשהוא מצביע בכל פעם על אות אחרת עם השהיה מסוימת. את זמן ההשהיה נקבע במשתנה TIME: המוגדר בהליך PLAY עוד בתחילת התכנית. TIME: הוא משתנה 'לא-מקומי' (Non-Local), ולכן הוא מוכר על-ידי הליכים ברמה נמוכה יותר בתכנית. מיותר היה להעביר את ערכו מהליך אחד לאחר, מכיון שערכו נשמר כפי שהוא, בכל זמן ביצוע התכנית.

PLY פונה ל-BDK ברגע שהמשתמש לוחץ על מקש כלשהו.

הגדרת BDK

```
TO BDK :X :L3 :OBJ
STCR 1 2
IF :X = 6 [IF FIRST :L3 = :OBJ [SMILY] [CRYLY] WAIT 50]
IF :X = 19 [IF FIRST BF :L3 = :OBJ [SMILY] [CRYLY] WAIT 50]
IF :X = 32 [IF LAST :L3 = :OBJ [SMILY] [CRYLY] WAIT 50]
END
```

ב-BDK בודקים אם ערך הקואורדינטה X:, בה היה הסמן בעת ההקשה, מצביע על המקום של האות הראשונה. אחרת, עורכים את הבדיקה עבור האות השניה ולבסוף - עבור האות השלישית. במשפט שבו ערכו של X: מקיים את התנאי, בודקים אם האות עליה מצביע הסמן זהה עם האות העליונה. בהתאם לתשובה, ההליך פונה ל-SMILY, או ל-CRYLY.

זה למעשה עיקר התכנית. לא מובאים כאן ההליך SMILY, או CRYLY, וגם לא ה"אותיות" המופיעות ברשימה, שאותם תוכל להגדיר בעצמך.

נציג לדוגמה שרטוט של אחת האותיות, האות "בית":

הגדרת BEIT

```
TO BEIT
TYPE "$$$$$
REPEAT 6 [AMUD]
STCR FIRST CURSOR (LAST CURSOR) - 5
TYPE "$$$$$$$
END
```

ההליך BEIT משתמש בהליך AMUD, אשר מדפיס את הסימן '$' ולאחר מכן מציב את הסמן מתחת לסימן שהודפס.

הגדרת AMUD

```
TO AMUD
TYPE "$ STCR (FIRST CURSOR) + 1 (LAST CURSOR) - 1
END
```

מומלץ לתת בתחילת התכנית הסברים והנחיות על דרך המשחק, להציג למשתמש הודעה בה הוא מתבקש להזין את מספר הפעמים שהוא רוצה לשחק, ואת זמן ההשהיה שהוא מעוניין בו.

<u>הערה:</u> בתכנית שהוצגה קיימים נוסף על המשתנה TIME: משתנים אחרים 'לא-מקומיים'. העברת ערכיהם להליכים ברמה נמוכה יותר היא

מיותרת בקטעים מסוימים, מכיון שהם לא משתנים בחלק זה של התכנית. נסה למצוא אותם על מנת לחסוך בהגדרת משתנים.

ההוראה REPEAT - הרחבות

כל הוראה שניתן לרשום במצב הרגיל של לוגו, יכולה להיות אחת מהוראות רשימת הקלט של REPEAT. כאשר ערכו של משתנה הוא שם של הליך, ניתן באמצעות ההוראה REPEAT לבצע את ההוראות הכלולות בו.

למשל, לא נוכל לכתוב הוראה כמו ---> ?4 + 10 / 2

וגם בתוך REPEAT, אין זה אפשרי.

אם נכתוב ---> ?REPEAT 1 [4 + 10 / 2]

נקבל הערה: **I DON'T KNOW WHAT TO DO WITH 9**

REPEAT אינה פונקציה המחשבת ערך ולכן,

לא נוכל לרשום ---> ?PR REPEAT 1 [4 + 10 / 2]

עלינו לכלול ברשימת הקלט הוראות לביצוע, ולא חישובים המחזירים ערך, כנהוג עם פונקציות למשל.

משפט נכון יהיה לדוגמה ---> ?REPEAT 1 [PR 4 + 10 / 2]

דוגמה נוספת ---> ?REPEAT 1 [TYPE (SUM 3 4 5 6)]

לדוגמה נגדיר פונקציה הפועלת בשיטת הכתיב **הפולני** בהילוך **לאחור** (Reverse Polish Notation). בשיטה זו מציגים את פעולת החשבון בסוף הביטוי, אחרי שני הנתונים שהיא פועלת עליהם. למשל,

נתון הביטוי: 45 + 17

נכתוב אותו כך: 45 17 +

כזכור, בכתיב הפולני המקובל פעולת החשבון נרשמת בהתחלה, בצורה הבאה: + 45 17

נכתוב פונקציה RVRSPLSH, המקבלת שלושה קלטים (שני מספרים ואחת מפעולות החשבון) ומחשבת את ערך הביטוי.

הגדרת RVRSPLSH

```
TO RVRSPLSH :N1 :N2 :PEULA
IF :PEULA = "+ [OP :N1 + :N2]
IF :PEULA = "- [OP :N1 - :N2]
IF :PEULA = "* [OP :N1 * :N2]
IF :PEULA = "/ [OP :N1 / :N2]
END
```

ב-RVRSPLSH רשומים 4 משפטי תנאי כמספר פעולות החשבון. אם מספרם היה גדול יותר, היה צורך במספר רב יותר של משפטי תנאי.

שים לב, כי לא ניתן להשתמש במשפט 'OP :N1 :PEULA :N2'. הערך של :PEULA בביטוי זה הוא **הסמל** של פעולת החשבון, אך לא הפעולה עצמה. וכך, הקלט של OP הוא :N1 ולא ערך הביטוי. כפי שלמדנו, OP פועלת על ערך אחד בלבד.

גם המשפט 'OP (SE :N1 :PEULA :N2)' לא יענה על הדרישה. כאן, תחזיר OP רשימה המכילה את שלושת הערכים. לדוגמה:
אם 9 = N1: ,4 = N2: ו-"* = PEULA:, ערך הביטוי המחושב על-ידי הפונקציה SE הוא '[4 * 9]'. כלומר, OP מקבלת **רשימה** המכילה סמלים המרכיבים ביטוי חשבוני' ולא את ערכו של הביטוי החשבוני.

אם OP תקדים את הביטוי **בתוך** הרשימה, היא יכולה להעביר את ערך הביטוי למי שפנה אליה.
נרשום: (SE "OP :N1 :PEULA :N1)

על פי הדוגמה הקודמת, הרי זה כאילו רשמנו [OP 9 * 4]. כאן, הקלט של OP הוא ערך אחד 36, השווה לערך הביטוי '4 * 9'.

REPEAT מפעילה הוראות המופיעות בתוך רשימה, ולכן נוכל להשתמש בה בהגדרת RVRSPLSH2.

הגדרת **RVRSPLSH2**

```
TO RVRSPLSH2 :N1 :N2 :PEULA
REPEAT 1 (SE "OP :N1 :PEULA :N2)
END
```

הפונקציה **WORD** יכולה לפעול על יותר מאשר שני נתונים, אם נתחום את כל ההוראה בסוגריים עגולים.

לדוגמה, (WORD "CO "CA "_ "CO "LA)

לדוגמה, נתונה רשימת המספרים [7 23 360 44] כערך של המשתנה RS:, עלינו להרכיב ממספרים אלה מספר אחד. לא נוכל לרשום את הביטוי 'WORD :RS', כי WORD פועלת על ערכים ולא על רשימות. אבל, באמצעות הפונקציה SE ניתן יהיה לצרף את המילה WORD" יחד עם איברי הרשימה לרשימה אחת.

לדוגמה, הביטוי 'SE "WORD :RS'

יהיה שווה ל- [WORD 7 23 360 44]

הביטוי שברשימה עדיין לא תקין לחיבור כל האיברים למילה אחת, כי חסרים בו הסוגריים העגולים.

לכן נתקן זאת בדרך הבאה: (SE "("WORD :RS "))

הפעם נקבל את הרשימה הבאה: [(WORD 7 23 360 44)]. מכיון שהביטוי להרכבת המילה מופיע בתוך רשימה, נוכל להשתמש במשפט המכיל את הצמד OP/REPEAT לקבלת הערך של הביטוי.

הגדרת **ONEWORD**

```
TO ONEWORD :RS
REPEAT 1 (SE "OP "( "WORD :RS ") )
END
```

כאשר נרשום ---> ?PR ONEWORD [1 23 456 7890]

נקבל: 1234567890

REPEAT איננה פונקציה, ולכן אין המשפט המורכב מהוראה זו יכול לשמש כחלק מביטוי חישובי, או כערך למשתנה כלשהו. תפקידה העיקרי של REPEAT הוא **בביצוע חוזר** של סידרת הוראות בלולאה. להפעלת הוראות הנתונות בתוך רשימה והפקת פלט מהם, משתמשים בפונקציה RUN.

הפונקציה RUN

RUN היא פונקציה אשר הקלט שלה הוא **רשימה**, והפלט הוא תוצאת החישוב על-פי הרשימה. במילים אחרות, היא קוראת את הכתוב ברשימה, ומבצעת את ההוראות שהיא קוראת.

תוכן רשימת קלט של RUN יכול להיות

- הוראות לביצוע
- ביטוי חישובי

א. הרשימה יכולה להכיל הוראות או הליכים, כפי שראינו זאת בהוראה REPEAT.

ז"א, התוצאה של RUN [רשימת הוראות]

שווה לתוצאה של - REPEAT 1 [רשימת הוראות]

לדוגמה נוכל לרשום ---> ?RUN [PR [I LOVE PARIS]]

וגם ---> ?RUN [DOLLARUP]

ב. RUN היא פונקציה, ועל-כן היא יכולה לפעול גם על ערך המוחזר מביטויים חישוביים. יש להורות למחשב מה לעשות בערך שחושב על-ידי RUN **עצמה**, בדומה לפונקציה הבאה:

```
TO OPEXP :LST
REPEAT 1 (SE "OP :LST)
END
```

הפונקציה OPEXP מקבלת כקלט למשתנה LST: רשימה, המכילה ביטוי חישובי, למשל [SQRT 81]. ערך הפונקציה OPEXP הוא תוצאת החישוב של הביטוי המופיע ברשימה, שבמקרה שלנו שווה ל-9.

OPEXP היא פונקציה, ולכן יש לפנות אליה באמצעות הוראה כלשהי הפועלת עליה. לדוגמה,

נכתוב ---> ?PR OPEXP [SQRT 81]

ונקבל: 9

נוכל לומר שהמשפט הזה קיים:

OPEXP [ביטוי חישובי] = RUN [ביטוי חישובי]

לדוגמה ---> ?PR RUN [4 + 10 / 2]

וגם ---> ?PR RUN [SQRT 81]

אם הערך של המשתנה :FUN הוא 'SQRT', וב-:MIS יש הערך '81',
נוכל לכתוב את הביטוי הבא: RUN (LIST :FUN :MIS)

<u>שים לב</u> - הרשימה [SQRT 81] אינה שווה לרשימה [:FUN :MIS].

נגדיר פונקציה RMV.FIRST המקבלת במשתנה :LIST רשימת נתונים, ובמשתנה :PRED - פרדיקט מוגדר הפועל על קלט אחד בלבד. פרדיקט זה יכול להיות פרדיקט המוגדר בשפה (כמו NUMBERP, LISTP, WORDP), או פרדיקט שהגדרנו בעצמנו. על הפונקציה להחזיר את הרשימה ללא האיבר הראשון המקיים את הפרדיקט.

הגדרת RMV.FIRST

```
TO RMV.FIRST :LST :PRED
IF EMPTYP :LST [OP []]
TEST RUN (LIST :PRED WORD "" FIRST :LST)
IFT [OP BF :LST]
IFF [OP FPUT (FIRST :LST) (RMV.FIRST BF :LST :PRED)]
END
```

נרשום את ההוראה ---> ?PR RMV.FIRST [MON 29 FEV 88] "NUMBERP

בהוראה זו הרשימה '[MON 29 FEV 88]' מהווה ערך ל-:LST, והפרדיקט 'NUMBERP' הוא הערך של :PRED.

נקבל את הפלט הבא: MON FEV 88

תוצאת החישוב ש-RUN מבצעת היא ערכו של ביטוי לוגי, הקובע אם האיבר הראשון של LST: הוא מספר (NUMBERP).
RUN פועלת על רשימה, לכן LIST בונה את הביטוי הלוגי ברשימה.

נתון המשפט: (LIST :PRED WORD "" FIRST :LST)
במשפט זה LIST מקבלת שני דברים:
- הערך של המשתנה PRED:
- ערך הביטוי WORD "" FIRST :LST

כידוע, **מילה** בלוגו צריכה להיות עם גרשיים בתחילתה, אחרת לוגו תזהה אותה כהליך.
לכן באה WORD ומחברת גרשיים לערך 'FIRST :LST'.
לדוגמה,
במחזור הראשון ערך הביטוי: (LIST :PRED WORD "" FIRST :LST)
הוא: [NUMBERP "MON]

WORD בנתה מילה המורכבת מהגרשיים ומהאיבר הראשון שברשימה, ובכך אפשרנו ללוגו לבצע את החישוב ומנענו שגיאה תחבירית.

* * * * *

תרגילים (המשך)

16. הגדר פונקציה המקבלת רשימת מספרים, ומחזירה אותן כמילה אחת ללא **הסיפרה** הראשונה. על הפונקציה להיות מורכבת ממשפט אחד בלבד.

17. הגדר פונקציה FUNWRD אשר יש לה שני משתנים: FUN: ו-LWRD:. המשתנה FUN: מקבל שם של פונקציה הפועלת על מילים (כמו RVRS, LENGTH, PIRUK). המשתנה LWRD: הוא רשימת מילים. הפונקציה FUNWRD מחזירה רשימה ובה תוצאת פעולת FUN: על כל אחד מאיברי הרשימה.

אם לדוגמה,

נרשום ---> ?PR FUNWRD "RVRS [ROSE DAHLIA TULIP]

נקבל את הפלט הבא: ESOR AILHAD PILUT

18. באחד התרגילים בשיעור קודם, היה צריך להגדיר פונקציה המחזירה רשימה של זוגות הערכים X ו-Y, בתחום מסוים עבור הפונקציה הקבועה ($y=x^2-x-1$).

הפעם, הגדר פונקציה FUNCXY, אשר מחשבת את הערכים של פונקציה מוגדרת כלשהי FUN: עבור ערכי X מ-'5-' עד '5' בהפרשים של 1, ומחזירה כפלט את רשימת זוגות הערכים X ו-Y.

נניח שקיימת הפונקציה FX3 לחישוב הפונקציה הבאה '$F(x)=x^3+2x-1$' כאשר 'FX3' מוגדרת במחשב כך:

```
TO FX3 :X
OP :X * :X * :X + 2 * :X - 1
END
```

אם נרשום ---> ?PR FUNCXY -5 "FX3

נקבל: [-5 -136][-4 -73] ... ... [5 134]

<u>שים לב</u>, במקום 'FX3' נוכל לרשום כל פונקציה מוגדרת אחרת הדורשת קלט אחד, כמו SQRT, SIN או ABS.

19. אחת השיטות למציאת פתרון למשוואות היא שיטת החציה. השתמש בשיטה זו להגדרת תכנית המקבלת פונקציה כלשהי למשתנה :FUNC, ומחשבת פתרון מקורב בדיוק של 0.0005.

<u>הדרכה:</u> מצא תחום בו קיים הפתרון. תחום זה נמצא בין שני ערכי X שעבורם ערכי הפונקציה הם בעלי סימנים מנוגדים:

f(x1)*f(x2) < 0

תחום זה ישמש כקטע התחלתי לחישוב:

א. יש למצוא את נקודת האמצע: m=(x1+x2)/2

ב. אם f(x1) * f(m) < 0 אזי המשתנה x2 יקבל את הערך של m,
אחרת, x1 יקבל את הערך של m.

ג. תנאי ההפסקה יהיה כאשר ABS f(m) < 0.0005

* * * * *

ההוראה THROW "TOPLEVEL

בתכניות הידברותיות כמו בלומדה או במשחק, רצוי לתת למשתמש בשלב מסוים את האפשרות להמשיך בתכנית או לסיימה. כלומר, יש לשאול אותו על כוונותיו ולקבל תשובה כן/לא ובהתאם לכך לנהוג. למשל:

DO YOU WANT TO CONTINUE ? (Y/N)

THROW "TOPLEVEL היא הוראה לסיום ביצוע של התכנית בכל מקום שהוא וחזרה ללוגו. ההוראה STOP למשל, מסיימת **רק** את ההליך בו היא מופיעה. לכן, כתנאי לסיום התכנית נוכל לרשום את המשפט הבא:

```
IF RC = 'N' [THROW "TOPLEVEL]
```

* * * * *

פרויקט:

דוגמה ליישום בחישובים עסקיים

תכנית זו משמשת להכנת חשבוניות בחנות לרהיטי גן הסוחרת בחמישה פריטים. תחילה היא מציגה על המסך טופס חשבונית ריק המוכן לקבלת נתונים לחישוב. עם סיום קבלת כל הקלט, היא ממלאת את המקומות המתאימים בתוצאות של החישובים הדרושים.

הגדרת HESHB

```
TO HESHB
MASACH
HADPES HISHUV 5 4
PR []
TYPE [ANOTHER HESHBONIT? (Y/N) : ]
IF RC = "Y [HESHB] [PR []]
END
```

HESHB מציגה את טופס החשבונית הריק באמצעות ההליך MASACH. לאחר מכן היא פונה ל-HADPES להדפסת תוצאת החישוב המתקבל מ-HISHUV. בתום הצגת החשבונית המוכנה, היא מאפשרת למשתמש להכין חשבונית חדשה:

```
TYPE [ANOTHER HESHBONIT? (Y/N) : ]
```

הגדרת MASACH

```
TO MASACH
CLEARTEXT
STCR 1 1 TYPE "HARAHIT
STCR 1 12 TYPE "KAMUT
STCR 1 20 TYPE "MHIR.YHIDA
STCR 1 32 PR "STCRHUM
REPEAT 38 [TYPE "=] PR [] PR []
PR "SHIMSHIA PR []
PR "SHULHAN PR []
PR "KISSE.NOAH PR []
PR "KISSE PR []
PR "NADNEDA PR []
REPEAT 38 [TYPE "=] PR []
STCR 16 18 PR "SACH.HACOL
STCR 18 18 (PR "15% "MAAM)
STCR 19 18 REPEAT 20 [TYPE "_]
STCR 21 18 PR "LATASHLUM
END
```

MASACH משתמשת בהליך STCR המוגדר כקיצור ל-SETCURSOR.

לאחר ביצוע MASACH, נקבל את הפלט הבא:

```
 HARAHIT    KAMUT   MHIR.YHIDA  STCRHUM
========================================
SHIMSHIA
SHULHAN
KISSE.NOAH
KISSE
NADNEDA
========================================
                 SACH.HACOL
                 15% MAAM
                 ____________________

                 LATASHLUM
```

הגדרת HISHUV

```
TO HISHUV :N :Y
IF :N < 1 [OP []]
STCR :Y 14
OP FPUT SCHUM (KELET RC 1) HISHUV :N - 1 :Y + 2
END
```

HISHUV מקבלת את הכמות עבור כל פריט באמצעות הפונקציה **KELET**. היא מחשבת את הסכום לכל אחד מחמשת הפריטים באמצעות הפונקציה **SCHUM**, ומציבה סכום של כל פריט ברשימה. מאוחר יותר נראה כיצד משתמשת הפונקציה SCHUM ברשימת מחירים נתונה, אשר מוגדרת בתת-פונקציה MEHIR לקליטת מחיר המוצר המתאים.
בתום הביצוע של MASACH מוצב הסמן מתחת לעמודה KAMUT בשורה של הפריט הראשון, ועם קבלת הנתון הוא עובר לשורה של הפריט הבא עד לקבלת כל הנתונים.

הגדרת KELET

```
TO KELET :RDL :N
IF 13 = ASCII :RDL [OP " ]
IF OR :RDL = 0 :N < 1 [OP :RDL]
OP WORD :RDL KELET RC :N - 1
END
```

KELET מגבילה את המשתמש לשתי הקשות בלבד. פעם אחת בקריאה רגילה ופעם שניה בקריאה רקורסיבית. ערך הפונקציה הוא מילה המורכבת מהתווים שהיא קולטת. אם באחד משני המחזורים הוקש <RET>, ערך הפונקציה של אותו מחזור הוא המילה הריקה. התוצאה של KELET משמשת קלט עבור SCHUM.

הגדרת SCHUM

```
TO SCHUM :WRD
IF EMPTYP :WRD [OP 0]
IF :WRD = 0 [OP 0]
IF NOT NUMBERP :WRD [OP SCHUM KELET RC 1]
TYPE :WRD
STCR ( FIRST CURSOR ) 23
TYPE MEHIR
OP :WRD * MEHIR
END
```

SCHUM מחשבת את הסכום עבור כל פריט, אם ערך הקלט הוא המילה הריקה, הרי הסכום הוא אפס. כך גם כאשר ערך הכמות הוא אפס.

- אם ערך הקלט איננו מספר, מתעלמת SCHUM מהערך שנקלט ומבצעת שוב פניה ל-KELET לקבלת קלט מתאים.
- אם הקלט הוא מספר, מודפסים הכמות והמחיר, המחושב על-ידי MEHIR. ערך הפונקציה במקרה זה הוא 'כמות * המחיר'.

הגדרת MEHIR

```
TO MEHIR
OP ITEM :N [950 160 210 280 250]
END
```

MEHIR מחזירה את האיבר ה-N: מרשימת המחירים.

הפונקציה HISHUV בונה רשימה שכל אחד מאיבריה מהווה את הסכום עבור אחד מחמשת הפריטים ולכן היא מתבצעת בחמישה מחזורים. ערכו של N: הוא בתחילה '5', ובכל קריאה מופחת ממנו אחד. MEHIR היא פונקציה ברמה נמוכה יותר, אשר משתמשת בערך של N: המוגדר בפונקציה HISHUV.

שים לב כי הספירה מתבצעת לאחור, ולכן המחירים מסודרים בהתאם בתוך הרשימה.

הרשימה המתקבלת על-ידי HISHUV מהווה קלט עבור ההליך HADPES אשר מדפיס את הסכומים, מחשב ומדפיס את התוצאה לתשלום.

הגדרת HADPES

```
TO HADPES :LST
PLTOUT :LST 4
STCR 16 32
TYPOUT RUN (SE "( "SUM :LST ") )
END
```

ההליך HADPES פונה לפונקציה **PLTOUT** להדפסת הסכום עבור כל פריט. לאחר מכן היא פונה לפונקציה **TYPOUT** ומעביר לה קלט שהוא מחיר הקניה המחושב על-ידי הביטוי

```
RUN (SE "( "SUM :LST ") )
```

הגדרת PLTOUT

```
TO PLTOUT :L :Y
STCR :Y 32
IF EMPTYP :L [STOP]
TEST FIRST :L = 0
IFT [TYPE "\ \ \-]
IFF [TYPE PRET FIRST :L]
PLTOUT BF :L (2 + :Y)
END
```

PLTOUT מדפיסה בעמודה של הסכום את סכום הקניה לכל פריט. אם הסכום שווה לאפס עבור פריט מסוים, היא מדפיסה קו '-' באותו מקום. אחרת, היא מדפיסה את הסכום ביישור סיפרת האחדות, באמצעות הפונקציה PRET.

הגדרת PRET

```
TO PRET :N
IF 4 < COUNT :N [OP :N]
OP PRET (WORD "\  :N)
END
```

PRET מחזירה את הקלט שלה, אשר מורכב מ-5 תווים. כל עוד מספר התווים בקלט קטן מ-5, היא מצרפת תו רווח בצד השמאלי.

הגדרת TYPOUT

```
TO TYPOUT :TTL
TYPE PRET :TTL
STCR 18 32
TYPE PRET INT :TTL * 0.15
STCR 21 32
PR PRET INT :TTL * 1.15
END
```

TYPOUT מקבלת למשתנה TTL: את ערך הסכום הכולל של הקניה. תחילה היא מדפיסה אותו, לאחר מכן היא מדפיסה את חישוב המע"מ ולבסוף - את הסכום לתשלום.

כך מתקבל טופס חשבונית מוכן למסירה ללקוח. אם רוצים לקבל את החשבונית במדפסת, הדבר ניתן על-ידי לחיצה על 'Shift+PrtSc', בעת שהטופס מוצג על המסך.

* * * * *

תרגילים (המשך)

20. כתוב תכנית לתרגול פעולות החשבון. התכנית תבחר באופן אקראי שני מספרים בתחום (1..99) ואת פעולת החשבון. התכנית תציג את השאלה בצורה נאותה, תבדוק אם תשובתו של התלמיד אכן שווה לתוצאת החישוב, ותגיב בהתאם.
ערוך את התרגיל בצורה נאה על המסך.

21. כתוב תכנית המציגה למשתמש רשימה של דברים שיש בהם מן המשותף, ורק איבר אחד בה יוצא דופן. למשל, רשימה של מדינות באירופה, וביניהן מדינה אחת השייכת לאסיה. אפשר להציג לדוגמה רשימה של רהיטים וביניהם שם של פרי כלשהו.

על המשתמש לגלות מהו האיבר יוצא הדופן. הוא יכול לעשות זאת על-ידי הקשת המספר המציין את מקומו הסידורי של אותו איבר ברשימה, או על-ידי איות השם. התכנית בודקת את התשובה ומגיבה בהתאם.

תוכל להגדיר רשימה הכוללת כמה רשימות כאלה (למשל 4 רשימות הכלולות ברשימה אחת), והתכנית תציג את הרשימות זו אחר זו.

אפשר להרחיב את הפתרון ולתת דיווח על מספר התשובות הנכונות מתוך השאלות שנשאלו.

22. כתוב תכנית הבוחרת מספר אקראי כלשהו בן 3 ספרות, ומבקשת מן המשתמש לנחש אותו. כל סיפרה במספר תופיע פעם אחת בלבד, ולמשתמש יינתן מספר מוגבל של ניחושים.

מהלך התכנית דומה בעיקרו למשחק 'בול וקליעה', שבו אחרי כל ניחוש יינתן רמז בדבר התוצאה:

א. עבור **כל סיפרה** נכונה בערכה, אך אינה מופיעה במקומה הנכון, יוצג התו 'K' המרמז על קליעה.

ב. עבור **כל סיפרה** נכונה גם מבחינת מקומה המדויק במספר, יוצג התו 'B' המורה על ביל.

ג. אם הניחוש אינו מכיל אף לא סיפרה נכונה אחת, תוצג ההודעה:
NO CORRECT DIGITS

בתום המשחק הצג הודעה על מספר הניחושים השגויים.

תן אפשרות למשתמש לשנות את ההגבלה של מספר הניחושים, כמו למשל על-ידי ההודעה הבאה שתוצג על המסך:

YOU WILL BE GIVEN 20 GUESSES
DO YOU WISH TO CHANGE THE LIMIT? (Y/N)

אם הוסכם שהמשתמש ישנה את המגבלה, הוא יתבקש להדפיס את מספר הניחושים הרצוי:

ENTER THE MAXIMUM NUMBER OF GUESSES YOU DESIRE

23. כתוב תכנית היוצרת מעבד תמלילים פשוט בעברית, אשר האותיות מתוכננות בו בשיטת הגרפיקה למחצה. מספר האותיות בשורה יהיה מוגבל לחמש, ומספר השורות במסך יהיה 3.

הנחייה:

- הגדר הליך לכל אות. אם המשתמש יקיש את האות 'ב' למשל, תופיע הצורה של האות הזו על המסך. תן למשתמש את האפשרות להדפיס רצף של אותיות בשורה.
- הגדר 'סמן' שיורה על מקום ההדפסה הבאה.

- אם 'הסמן' מגיע לסוף שורה, על התכנית להעביר אותו לשורה חדשה.
- תן אפשרות למשתמש לעבור לשורה חדשה אם רצונו בכך.
- תן אפשרות לניקוי מסך.

24. כתוב תכנית לחישוב והדפסה של תלוש משכורת לעובד במפעל מסוים. על התכנית לקבל מן המשתמש את הפרטים הבאים:
1) מס' זהות.
2) שם העובד.
3) מס' שעות העבודה.
4) השכר לשעה.

התכנית תבצע בדיקת תקינות הקלט, תדפיס אותו בתלוש, תחשב ותדפיס את המשכורת ברוטו, את מס ההכנסה, והמשכורת נטו.

המס שמשלם העובד מחושב באופן הבא:
- עבור הכנסה קטנה מ-800 ש"ח - 0%.
- עבור חלק ההכנסה מ-801 עד 1200 ש"ח - 25%.
- " " " מ-1201 עד 1800 ש"ח - 35%.
- " " " שלמעלה מ-1800 ש"ח - 50%.

25. כתוב תכנית כללית להכנת חשבונית-מס שתכיל עד 10 פריטים כלשהם. יש להגדיר בתכנית רשימה ובה כל הפריטים (למשל 30 פריטים) הנמכרים בחנות מסוימת. על התכנית לקבל מן המשתמש את שם הפריט והכמות לאותו פריט ולהדפיסם בחשבונית.

באמצעות פונקציית חיפוש יש לשלוף המחיר של פריט מתוך הרשימה ולהדפיסו. בחשבונית יש לחשב ולהדפיס:-
- את המחיר של כל יחידה ואת המחיר הכולל לפריט,
- את סך הכל לחשבונית ללא מע"מ,
- את סכום המע"מ,
- את הסכום לתשלום.

נספח א'

כתיבה בעברית

עד עתה הצגנו את שפת לוגו הפועלת באנגלית. את ההודעות ואת הדו-שיח של המשתמש עם המחשב הצגנו בשפה האנגלית בלבד. בכמה מקרים, הצגנו תווים בעברית בשיטת הגרפיקה למחצה.

במחשבים ניתן להציג מלל בעברית ולהקיש תווים בעברית. נלמד עתה את השיטות לעשות זאת במחשבי IBM ו-APPLE.

הדפסת תווים בעברית למשתמשי IBM LOGO

א. הטען מה-DOS את התכנית HEBREW ואחר-כך טען את התכנית LOGO. אם אתה נמצא בלוגו, וברצונך לעבור ל-DOS רשום את ההוראה '.DOS' (נקודה והמילה DOS ללא רווח). כדי לעבור חזרה ללוגו יש לרשום 'LOGO'.

ב. ברירת המחדל היא שמקש <CAPS> **נעול**, כלומר לא מגיב, והכתיבה היא באותיות גדולות באנגלית בלבד. אם נרצה לדעת מה מצב מקש זה נוכל לרשום:

?PR CAPS

אם הוא נעול נקבל:

TRUE

השלב הראשון הוא שחרור הנעילה של מקש CAPS.
על מנת לשחרר אותו, נרשום את ההוראה: SETCAPS "FALSE

כאשר המקש <CAPS> משוחרר, הוא יגיב על כל לחיצה ויאפשר לכתוב אותיות קטנות באנגלית. בלחיצה נוספת על <CAPS> נשוב לאותיות גדולות באנגלית.

ג. על מנת לכתוב בעברית, יש לעבור לצדו הימני של המסך, לכיוון ימין-שמאל. זאת עושים על-ידי לחיצה בו-זמנית על שני המקשים <ALT> ו-<TAB>. הסמן יופיע בצדו הימני של המסך. כדי לכתוב בעברית יש ללחוץ על <CAPS>. ניתן לעבור לצד שמאל של המסך עם לחיצה נוספת על <ALT> ו-<TAB>, או על-ידי מקש <RET>.

הדפסת תווים בעברית למשתמשי APPLE LOGO II

יש להביא את המתג הנמצא מתחת למחשב בצד שמאל למצב עברית. כשמקש <CAPS> יהיה לחוץ, תתאפשר הכתיבה בעברית.

פלט במדפסת

על מנת לקבל תדפיס על הנייר ולא על המסך, יש ליצור תחילה קשר בין המחשב למדפסת.

ב-IBM LOGO רושמים את הפקודה 'DRIBBLE "LPT1'. מעתה, נוצר הקשר וכל תדפיס יישלח למדפסת. וודא תחילה שהמדפסת מופעלת. לניתוק הקשר רושמים 'NODRIBBLE'.

כאשר עובדים בגרפיקה, כלומר במסך גרפי, ומעוניינים להדפיס את המסך, יש לפעול כך:

א. לטעון תחילה מה-DOS תכנית הנקראת **GRAPHICS**, ואחר-כך את תכנית LOGO.

ב. כאשר הציור המבוקש מוצג בשלמותו על המסך, יש ללחוץ על **Shift+PrtSc**. כל מה שמופיע על המסך הגרפי יודפס במדפסת.

ב-APPLE LOGO הקשר נעשה באמצעות הפקודה '.PRINTER 1'.
המספר '1' מציין את קו הקשר אל המדפסת. לניתוק הקשר רושמים '.PRINTER 0', הוראה שמחזירה אותנו למסך.

להדפסת המסך הגרפי, קיים סוג של מדפסת הפועל עם ההוראה הבאה:

```
(TYPE CHAR 9 "G CHAR 13)
```

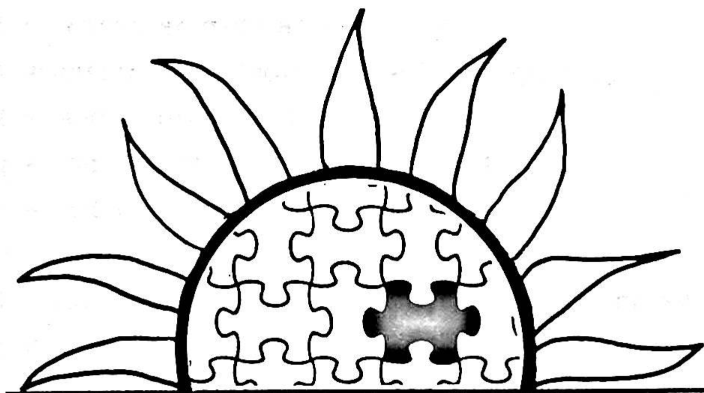
כאשר הציור מוצג על המסך והמחשב קשור למדפסת.
רצוי לכלול הוראה זו בהליך PRNTR שנגדיר ונשמור בדיסקט.

הגדרת PRNTR

```
TO PRNTR
.PRINTER 1
(TYPE CHAR 9 "G CHAR 13)
.PRINTER 0
END
```

במקרה זה, כאשר ההליך PRNTR נמצא בזיכרון המחשב, די לרשום PRNTR כדי שהציור המופיע על המסך הגרפי יודפס במדפסת.

למשתמש ב-APPLE LOGO II מומלץ לעיין במדריך של השפה.



נספח ב'

מילון פקודות

מילון זה מכיל את כל הפקודות שבהן דנו בספר זה, וכולל בהמשך רשימה של פקודות נוספות. בנוסף על פקודות אלו קיימים בלוגו הליכים (פרוצדורות) ופונקציות הקשורים בעיבוד קבצים, בתכונות, בקבוצות ועוד. המשתמש המתקדם טוב יעשה, אם יפנה לספרות המקצועית המצורפת לגירסת לוגו שרכש, כדי להכיר וללמוד את כל פקודות הלוגו.

חלק מהפקודות פועלות על ערך אחד, אחרות פועלות על שני ערכים, ויש כאלו שאינן דורשות כל ערך שהוא.

להלן הסמלים המורים על סוג הערך הדרוש לפקודה:

c = תו כלשהו.

lst = רשימה (list).

w = מילה (word).

n = מספר (number).

lst\w\n = רשימה או מילה או מספר.

lgexp = ביטוי לוגי (logical expression).

lstmt = רשימת הוראות.

proc = שם של פרוצדורה או של פונקציה.

נתון = כל נתון שהוא.

הסמל () מורה על כך, שהפקודה פועלת גם על מספר רב של נתונים, אם תוחמים את כל המשפט בסוגריים עגולים. הסמל <> מציין שאין חובה לרשום את הפרמטר.

כאשר יש אפשרות לכתוב את הפקודה בקיצור, רשמנו את השם המלא בסוגריים.

פקודות בספר

AND lgexp1 lgexp2 ()	מחזירה ערך אמת, אם כל הביטויים הם בעלי ערך אמת.
ARCTAN n	פונקציית הטנגס.
ASCII c	מחזירה את ערך ASCII של התו c.
BEFOREP w1 w2	מחזירה ערך אמת, אם המילה w1 אמנם נמצאת לפני w2 על פי הסדר המילוני.
(BUTFIRST) BF lst\w\n	מחזירה את הרשימה lst ללא האיבר הראשון, או את המילה w ללא התו הראשון, או את המספר n ללא הסיפרה הראשונה.
(BUTLAST) BL lst\w\n	מחזירה את הרשימה lst ללא האיבר האחרון, או את המילה w ללא התו האחרון, או את המספר n ללא הסיפרה האחרונה.
CAPS	פונקציה המחזירה ערך "אמת" אם הלחיצה על מקש CAPS אינה גורמת לשינוי.
CATALOG	מציגה על המסך את כל שמות הקבצים שבדיסקט. (APPLE LOGO II ו-APPLE LOGO).
CHAR n	מחזירה את התו שערך ASCII שלו הוא n.
CLEARTEXT	מנקה את מסך התמליל.
COS n	פונקציית הקוסינוס.

COUNT lst\w\n	מחזירה את מספר איברי הרשימה lst, או את מספר התווים המרכיבים את המילה w, או את מספר הספרות המרכיבות את המספר n. (ב-APPLE LOGO היא פועלת רק על רשימה).
CURSOR	מחזירה את ערכי הקואורדינטות של הסמן במסך התמליל.
DIFFERENCE n1 n2	פונקציית ההפרש. (לא קיימת ב-APPLE LOGO)
DIR	מציגה על המסך את שמות כל הקבצים שבדיסקט עבור הגירסה של IBM LOGO.
.DOS	יציאה מלוגו אל מערכת DOS.
(EDIT) ED <"proc>	מעבירה את הפעילות למסך העריכה, עם אפשרות לקבלת הליך אחד או יותר בהתאם לנדרש.
EDITFILE "שם-קובץ"	מעבירה מן הדיסקט אל מסך העריכה את הקובץ הרשום בפקודה.
EMPTYP lst\w\n	בודקת אם הרשימה lst ריקה. פועלת גם על מילה ומספר.
END	מציינת סוף הגדרה של הליך.
EQUALP נתון1 נתון2	מחזירה ערך אמת, אם שני הנתונים שווים.
(ERASE) ER "proc	מוחקת מן הזיכרון את ההגדרה הנתונה.
ERALL	מוחקת את כל ההגדרות מן הזיכרון.

ERASEFILE "שם-קובץ	מוחקת מן הדיסקט את הקובץ הנתון.
ERPS	מוחקת את כל הפרוצדורות והפונקציות שבזיכרון.
FIRST lst\w\n	מחזירה את האיבר הראשון של הרשימה lst. פועלת גם על מילה ומספר.
FPUT נתון lst	מציבה את הנתון בתחילת הרשימה lst.
IF lgexp lstmt1 <lstmt2>	אם ערך הביטוי lgexp הוא אמת, מתבצעת רשימת ההוראות lstmt1. אחרת, מתבצעת רשימת ההוראות lstmt2.
(IFFALSE) IFF lstmt	אם ערך הביטוי השמור על-ידי TEST הוא שקר, מתבצעות ההוראות שב-lstmt.
(IFTRUE) IFT lstmt	אם ערך הביטוי השמור על-ידי TEST הוא אמת, מתבצעות ההוראות שב-lstmt.
INT n	מחזירה את הערך השלם של n.
ITEM n lst\w\n	מחזירה את האיבר ה-n שברשימה lst. פועלת גם על מילה ומספר (לא ב-APPLE LOGO).
KEYP	מחזירה ערך אמת אם הוקש מקש כלשהו.
LAST lst\w\n	מחזירה את האיבר האחרון שברשימה lst. פועלת גם על מילה וגם על מספר.
() LIST 1נתון 2נתון	מציבה את נתוני הקלט בתוך רשימה שהיא בונה.

LISTP נתון	מחזירה ערך אמת אם הנתון הוא רשימה.
LOAD "שם-קובץ	טוענת מן הדיסקט את הקובץ המבוקש.
LPUT נתון lst	מציבה את הנתון בסוף הרשימה lst.
MEMBERP נתון lst\w\n	מחזירה ערך אמת אם הנתון הוא - - איבר ברשימה lst, - תו במילה w, - או סיפרה במספר n. (ב-APPLE LOGO היא פועלת רק על רשימה)
NOT lgexp	מחזירה ערך אמת אם ערך הביטוי lgexp הוא שקר.
NUMBERP נתון	מחזירה ערך אמת אם הנתון הוא מספר.
(OUTPUT) OP נתון	מחזירה את תוצאת החישוב של הנתון.
OR lgexp1 lgexp2 ()	מחזירה ערך שקר אם כל ערכי הקלט הם שקר.
PO "proc	מציגה על המסך את ההגדרה המבוקשת.
PONS	מציגה על המסך את כל שמות המשתנים וערכיהם.
POPS	מציגה על המסך את ההגדרות של כל הפרוצדורות והפונקציות שבזיכרון.
POTS	מציגה את שמות כל ההליכים והפונקציות שבזיכרון.

POWER n1 n2 מחזירה את הערך של n1 בחזקת n2.

() נתון PR (PRINT) מדפיסה את ערך הנתון על המסך.

PRODUCT n1 n2 () מחזירה את מכפלת n1 ב-n2.

QUOTIENT n1 n2 מחזירה את הערך של n1 חלקי n2. ב-APPLE LOGO היא מחזירה את המנה של n1/n2.

RANDOM n מחזירה מספר אקראי בתחום (n-1 ... 0).

(READCHAR) RC מצפה להקשת מקש, ומחזירה את ערכו.

REMAINDER n1 n2 מחזירה את השארית של n1/n2.

REPEAT n lstmt מבצעת n פעמים את ההוראות שברשימה lstmt.

(READLIST) RL מצפה לקלט מן המשתמש ומחזירה אותו כרשימה לאחר הקשת <ret>.

ROUND n מעגלת את הנתון n.

RUN lstmt גורמת לביצוע ההוראות שברשימה lstmt.

SAVE "שם-קובץ" שומרת את כל ההגדרות שבזיכרון המחשב בקובץ דיסקט בעל השם הנתון.

(SENTENCE) SE נתון1 () נתון2 מחזירה את ערכי הנתונים ברשימה אחת.

SETCURSOR [y x] מציבה את הסמן בנקודה (y,x).

SETWIDTH n	קובעת n עמודות בשורה הפיזית במסך.
נתון SHOW	מציגה על המסך את הנתון.
SIN n	פונקציית הסינוס.
SQRT n	מחזירה את השורש הריבועי של a.
STOP	מסיימת ביצוע של ההליך.
SUM n1 n2 ()	מחזירה את ערך סכום הקלטים.
TEST lgexp	שומרת את ערך הביטוי הלוגי lgexp.
THROW "TOPLEVEL	עוצרת את ביצוע התכנית ומחזירה ל-LOGO.
TO w	מכריזה על הגדרת פרוצדורה או פונקציה חדשה.
נתון TYPE ()	מדפיסה את הנתון על המסך. הסמן נשאר אחרי התו האחרון שהודפס.
WAIT n	הוראת השהייה של n יחידות זמן.
WIDTH	פונקציה המחזירה את מספר העמודות הקיים ברגע זה בשורה הפיזית במסך.
WORD w1 w2 ()	מחברת את הקלטים למילה אחת.
נתון WORDP	מחזירה ערך אמת אם הנתון הוא מילה.

פקודות נוספות בלוגו

בלוגו קיים מספר רב של פקודות, אשר לא דנו בהן בספר זה. נציג כאן חלק מהן.

פקודות למשתנים

MAKE "var נתון	הוראה המציבה את הנתון כערך למשתנה .var
THING "var	מתנהגת כמו הנקודתיים. לדוגמה, אם הערך של NUM הוא 6, נוכל לרשום: PR THING "NUM במקום: PR :NUM
NAME נתון "var	הוראה המציבה את ערך הנתון למשתנה .var
NAMEP "w	פרדיקט הבודק אם w הוא משתנה בעל ערך.
ERN "var	מחיקת המשתנה .var
ERNS	מחיקת כל המשתנים המוגדרים במחשב.
EDNS	הגדרת משתנים במסך העריכה.

פקודות להליכים

TEXT "proc	פונקציה שהפלט שלה הוא רשימת ההוראות של הפרוצדורה .proc
DEFINE "proc [list]	הוראה להגדרת הליך בשם proc, אשר list מהווה את רשימת ההוראות שלו. פקודה זו מאפשרת הגדרת הליכים מתוך הליך אחר.

DEFINEDP "w — פרדיקט הבודק אם w הוא הליך המוגדר במחשב.

PRIMITIVP "w — פרדיקט הבודק אם w היא פקודה השייכת ללוגו.

COPYDEF "oldw "neww — הוראה להעתקת ההוראות של ההגדרה old להגדרה אחרת new. ההוראה מתאימה להגדרת שמות מקוצרים עבור פקודות השייכות ללוגו.

פקודות לשטח העבודה

.CONTENTS — פונקציה להצגת כל המילים הידועות ללוגו. באמצעות הוראה זו תוכל לדעת מהן המילים השייכות לנוסח שבידיך.

NODES — פונקציה שהפלט שלה הוא מספר היחידות הפנויות בזיכרון המחשב.

RECYCLE — הוראה המנקה את תאי הזיכרון שכבר מלאו את תפקידם ומכשירה אותם לשימוש חוזר.

פקודות המתערבות בביצוע

PAUSE — פקודה הגורמת להפסקה זמנית של כל פעולה שהיא. בהפסקה זו ניתן לבצע **כל הוראה אחרת** בלוגו. ב-APPLE LOGO II וכן גם ב-APPLE LOGO אפשר להפסיק גם על-ידי לחיצה על ^Z. ב-IBM LOGO משתמשים במקש [F5].

CO (CONTINUE) ממשיכה את הביצוע אשר הופסק על-ידי PAUSE.

קיימת אפשרות לעצירה של פעולה כלשהי על-ידי לחיצה על **W^** במחשב APPLE, ולחיצה על **Ctrl+NumLock** ב-IBM LOGO. הקשה על כל מקש אחר תגרום להמשך הפעולה.

פקודות גרפיות

BG (BACKGROUND) פונקציה המציינת את צבע המסך.

BK n (BACK) הצב נע אחורה n צעדים.

CLEAN מנקה את המסך הגרפי. הצב נשאר במקום.

CS (CLEARSCREEN) מנקה את המסך הגרפי. הצב חוזר למקומו ההתחלתי

DOT [x y] הוראה לסמן נקודה בקואורדינטה הנתונה.

FD n (FORWARD) הצב מתקדם n צעדים.

FENCE מצב שבו הצב לא חורג מגבולות המסך.

FULLSCREEN העברה למסך גרפי מלא. **FS** - קיצור עבור גירסת IBM LOGO.

HEADING מחזירה את ערך כיוון הצב במעלות ביחס לצפון

HOME מציבה את הצב במקומו ההתחלתי מבלי לנקות את המסך.

(HIDETURTLE) HT	הצב לא נראה.
(LEFT) LT n	קובעת את כיוון הצב ב-n מעלות שמאלה ממצבו האחרון.
(MIXEDSCREEN) MS	לקבלת מסך מפוצל בגירסת IBM LOGO.
PEN	פונקציה המציינת את מצב העט וצבעו.
PENCOLOR	פונקציה לציון צבע העט.
(PENDOWN) PD	מצב שבו הצב משרטט בעת תנועתו.
(PENERASE) PE	מצב שבו הצב מוחק בעת תנועתו.
(PENREVERSE) PX	מציירת קווים על מסך נקי, ומוחקת קווים מצוירים.
POS	מחזירה את ערכי הקואורדינטות של הנקודה בה נמצא הצב.
(PENUP) PU	מצב שבו הצב אינו משרטט בעת תנועתו.
(RIGHT) RT n	קובעת את כיוון הצב ב-n מעלות ימינה ממצבו האחרון.
SETBG n	הוראה לקבוע את צבע המסך.
(SETHEADING) SETH n	קובעת את כיוון הצב ב-n מעלות ימינה ביחס לצפון.
SETPC n	הוראה לקבוע את צבע העט.
SETPEN [w n]	הוראה לקבוע את מצב העט וצבעו.

SETPOS [x y] — מניעה את הצבא עד לנקודה (x,y).

SHOWNP — פרדיקט המציין לנו אם הצב נראה או בלתי נראה.

.SCRUNCH — פונקציה המורה את היחס בין רוחב המסך לבין אורכו (ברירת המחדל היא 0.8).

.SETSCRUNCH n — הוראה לקבוע את היחס בין הרוחב לאורך.

SETX n — מניעה את הצב עד לעמודה n באותה שורה.

SETY n — מניעה את הצב עד לשורה n באותה עמודה.

SPLITSCREEN — קבלת מסך מפוצל עבור גירסות ה-APPLE.

(SHOWTURTLE) ST — מצב שבו הצב נראה.

TEXTSCREEN — קבלת מסך התמליל. TS קיצור עבור הנוסח של IBM LOGO.

TOWARDS [x y] — מחזירה את כיוון הצב במעלות ימינה מן הנקודה בה הצב נמצא אל הנקודה (x,y).

WINDOW — מצב שבו הצב חורג מגבולות המסך ונעלם.

WRAP — מצב שבו הצב 'עוטף' את המסך אם הוא חורג מגבולותיו.

XCOR — מחזירה את ערך הקואורדינטה x בה הצב נמצא.

YCOR — מחזירה את ערך הקואורדינטה y בה הצב נמצא.

מעבר בין סוגי מסכים

בלוגו, כל הוראה גרפית תעביר אותנו למסך הגרפי. מסך זה יהיה מחולק לשני חלקים, החלק העליון לציור, והחלק התחתון ובו מספר שורות שנועדו לכתיבת תמליל (הוראות). אבל, ניתן לעבור למסך גרפי שלם, שכולו מוקדש לציור.

לביצוע ההוראות שלהלן אפשר לכתוב את הפקודה במלואה, לכתוב את הקיצור (ב-IBM LOGO), או להקיש מקש פונקציונלי או זוג מקשים, כפי שניתן בטבלה הבאה:

א. לקבלת מסך תמליל לכתיבה בלבד (מסך העבודה):
ב-IBM LOGO רושמים TEXTSCREEN או בקיצור TS, או לוחצים על <F1>. ב-APPLE LOGO II וב-APPLE LOGO רושמים TEXTSCREEN, או לוחצים על (^T).

ב. לקבלת מסך מפוצל, גם לכתיבה וגם לציור:
ב-IBM LOGO רושמים MIXEDSCREEN או בקיצור MS, או מקישים <F2>.
ב-APPLE LOGO II וב-APPLE LOGO רושמים SPLITSCREEN, או לוחצים על (^S).

ג. לקבלת מסך גרפי שלם, לציור בלבד:
ב-IBM LOGO רושמים FULLSCREEN או בקיצור FS, או מקישים <F4>.
ב-APPLE LOGO II וב-APPLE LOGO רושמים FULLSCREEN, או לוחצים על (^L).

נספח ג'

MIT/TERRAPIN

בגירסת לוגו MIT/TERRAPIN (המצויה במחשבי APPLE ו-COMMODORE) יש פקודות אשר אינן זהות עם אלו השייכות לגירסה LCSI המופיעה בספר זה, למרות שהן מתנהגות באופן דומה. יש פקודות שנמצאות בגירסת LCSI ואינן כלולות בגירסת MIT.

פקודות מקבילות לגירסת LCSI

מטרת הרשימה שלהלן להנחות את המשתמש בגירסת MIT לבחור את הפקודות המתאימות לגירסת לוגו המוצגת בספר זה. נציג אותן כאן בהתאמה:

גירסת MIT/TERRAPIN	עמוד בספר	גירסת LCSI
ALLOF	146	AND
ATAN	96	ARCTAN
CATALOG	59	CATALOG <APPLE>
COUNT	192	COUNT <IBM>
---	239	CURSOR
EMPTY?	165	EMPTYP
EQUAL?	153	EQUALP
ER ALL	44	ERALL
ER PROCEDURES	44	ERPS
INTEGER	93	INT
ITEM	191	ITEM <IBM>
---	247	KEYP

LISTP	165	LIST?
LOAD	59	READ
MEMBERP	182	MEMBER?
NUMBERP	153	NUMBER?
OR	147	ANYOF
PONS	52	PO NAMES
POPS	44	PO PROCEDURES
.PRINTER <APPLE>	273	OUTDEV
PRODUCT	92	---
RC (READCHAR)	211	RC (READCHARACTER)
RL (READLIST)	228	RQ (REQUEST)
SETCURSOR	24	CURSOR
SUM	91	---
THROWW "TOPLEVEL	262	TOPLEVEL
TYPE	78	PRINT1
WAIT	25	---
WORDP	166	WORD?

פקודות עריכה:

^A	מציב את הסמן בתחילת השורה.
^B	מביא את הסמן לסוף השורה.
^P	מעתיק את שורת ההוראות האחרונה שניתנה למחשב.
^D	מוחק את התו שעליו עומד הסמן.
[DEL]	מוחק את התו שמשמאל לסמן.
[ESC]	כנ"ל.
^G	מפסיק כל עבודה המתבצעת על-ידי המחשב.
[<--]	מניע את הסמן שמאלה.
[-->]	מניע את הסמן ימינה.

פקודות הפועלות במסך העריכה בלבד:

^B מביא את הסמן לתחילת העריכה.
^F מביא את הסמן לסוף העריכה.
^N מניע את הסמן שורה אחת מטה.
^P מניע את הסמן שורה אחת מעלה.
^O פותח שורה חדשה בין שתי שורות.
^K מוחק מהסמן ימינה עד סוף השורה.
^C יציאה מן העורך עם הגדרה.
^G יציאה מן העורך עם הפסקת העבודה ללא הגדרה.

הכניסה למסך העריכה נעשית גם על-ידי המילה TO.

הבדלים בתחביר

נוסף על ההבדלים שהזכרנו, נמצא גם הבדלים בתחביר:

א. בפקודות הפועלות על הליכים כמו PO, ED ו-ER, כותבים את שם ההליך ללא הגרשיים.
במקום PO "MIUN למשל, כותבים PO MIUN.

ב. במשפטי התנאי, ההוראות אינן נמצאות בתוך סוגריים מרובעים.
לדוגמה, במקום - IF EMPTYP :LST [OP "TRUE] [OP "FALSE]
כותבים: IF EMPTY? :LST THEN <u>OP "TRUE</u> ELSE <u>OP :FALSE</u>

ניתן להשמיט את המילה THEN (וכמובן שהקו הינו לשם הדגשה בלבד).

הדבר נכון גם לגבי IFF ו-IFT. ההוראות נכתבות ללא הסוגריים המרובעים.

ג. ציון של נקודה על המסך בהוראה CURSOR לא מופיע בתוך סוגריים מרובעים. לדוגמה, במקום - SETCURSOR [30 10]
כותבים: CURSOR 30 10

הדבר נכון גם עבור ההוראה SETXY.

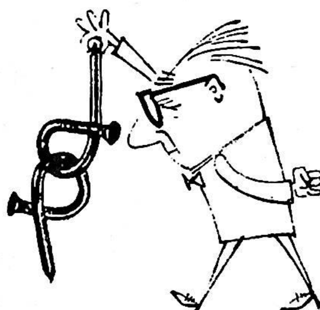
ד. נתון שלילי יש לכתוב בין סוגריים עגולים, אחרת יתייחס המחשב אל סימן המינוס כאל פעולת החיסור.
יש לכתוב - CURSOR 20 (-10)
ולא: CURSOR 20 -10

במחשב COMMODORE הקשר למדפסת נעשה עם המילה 'PRINTER' והניתוק - עם המילה 'NOPRINTER'.

נספח ד'

אינדקס תכניות דוגמה

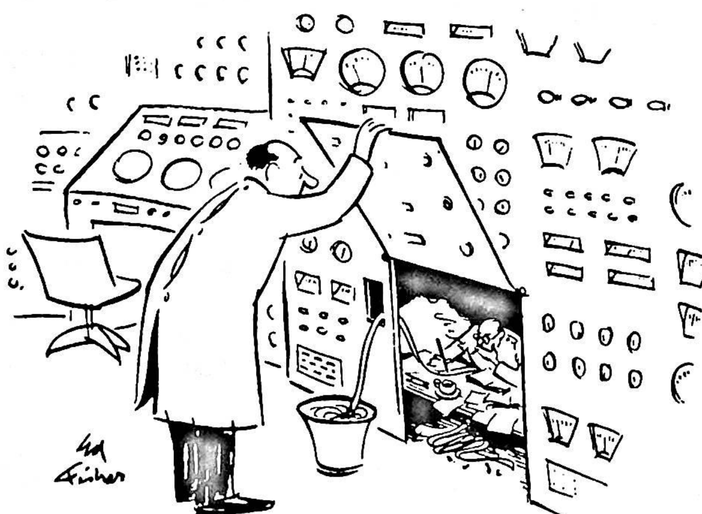
שמות ההגדרות המשמשות 'דוגמה', מובאים כאן בהתאם לסדר הופעתם בספר, ולא לפי סדר האלף-בית. כך, תמצא שהן מסודרות לפי רמת הידע הנרכש ככל שמתקדמים עם החומר.



לכל בעיה — פתרון !

נספח ה'

אינדקס הוראות



האם זהו כל הסוד !

נספח ו'

ביבליוגרפיה

לוגו בעברית:

1. אלקין רבקה, ירושלים במשיכת צב - רעיונות להתנסות ויצירה, אוני' בן-גוריון, 1988.

2. גבעון יהושפט, פרופ', סדרת הרפתקאות בלוגו, באג, 1987.

3. לירון אורי, נשר פרלה, צצקיס רינה, מדריך לעסקי צבים, באג, חלק א - 1984, חלק 2 - 1988.

4. -, צו לצב - עלון אגודת לוגו בישראל, מרכז לוגו - המחלקה להוראת הטכנולוגיה והמדעים בטכניון, החל מ-1986.

ספרות מומלצת באנגלית:

5. Abelson Harold, APPLE LOGO, Byte/McGraw Hill, 1982.

6. Abelson Harold and diSessa A.A., Turtle Geometry, MIT Press, 1984.

7. Harvey Brian, Computer Science LOGO Style, MIT Press, vol I - 1985, vol II - 1986, vol III - 1987.

8. Papert S., Mindstorms, Basic Books N Y , 1980.

9. Ross Peter, LOGO Programming for the IBM PC, Eddison Wesley, 1985.

לקורא המתעניין:

10. הראל דוד, פרקי יסוד במדעי המחשב, האוניברסיטה המשודרת משרד הבטחון ההוצאה לאור, 1985.

11. האן יחיאל ועמיהוד יצחק, המחשב האלקטרוני, הוצאת הוד-עמי, 1985 (אושר כספר לימוד).

LOGO - תכנות פונקציונלי מציג את שפת לוגו מנקודת ראות חדשה, השונה בתכלית מזו המוכרת לנו על ידי גרפיקת הצב. הוא מאיר תמונה מקיפה וברורה של הפוטנציאל הטמון בה, מציב ללומד אתגר ועוזר לו להתמודד עם תכנות ברמה גבוהה.

לוגו הינה גירסה ידידותית בעלת אופי פונקציונלי, הקרובה לשפת LISP לתחומי השימוש של בינה מלאכותית ומערכות מומחה. על-כן הספר עוסק בעיקר בתכנות הפונקציונלי, שבו הרכבת פונקציות הינה אמצעי עִיקרי ליצירת ביטויים מורכבים.

שיטת הלימוד מיישמת לקחים מנסיונה של המחברת. היא מנחה את הלומד מן הקל אל הכבד, מצעדים ראשונים בתכנות מתקדם ועד לניצול של העוצמה והתחכום של השפה. היא מדריכה כיצד לתכנת באופן מבוקר ומובנה כדי להגיע לרמת מיומנות גבוהה.

המחברת מדגישה נושאים המיושמים גם בשפות תכנות פונקציונליות אחרות: תכנות מבני ורקורסיבי, נוהלים לבניית פרוצדורות ופונקציות עיבוד רשימות ועוד. בספר כ-100 תכניות לדוגמה ופרויקטים מקיפים בתחומי פעילות שונים, וגם עשרות תרגילים לעבודה עצמית, המשולבים בנושאי הלימוד.

הספר מציג את גירסת IBM LOGO שפותחה על-ידי LCSI (Logo Computer Systems Inc). הוא מתאים גם למשתמשי APPLE LOGO II ו-APPLE LOGO. ניתן הסבר על גירסת MIT/TERRAPIN ל-קומודור.

LOGO - תכנות פונקציונלי מיועד לתלמידים בבתי הספר התיכוניים ובאוניברסיטאות, וללומד השקדן הרוצה להתמחות בשפה לוגית מתקדמת.

המחברת:

ש. מנולה הינה מורה למקצועות המחשב ומתמחה בשפת לוגו. הדריכה בקורס "הכרת המחשב וישומיו בחינוך" בסמינר לוינסקי, עוסקת בהוראה בבתי-ספר תיכוניים, ומלמדת את שפת לוגו בכיתות של תלמידים מחוננים.

ככל שלמדה להכיר את השפה, כן התעוררה סקרנותה לגלות את רזיה, והוסיפה להתפעל מכוחה ומן התחכום שבה. את הנסיון שהיא רכשה העלתה על הכתב בספר זה, ובחרה בדרכים הפשוטות, היעילות והבהירות ללימוד השפה, הבנתה והשימוש היעיל בה.

מסת"ב ISBN 965-361-003-1